EXHIBIT 1004

US007447791B2

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

(10) **Patent No.: US 7,447,791 B2**
(45) **Date of Patent: Nov. 4, 2008**

(54) **TRANSMISSION AND RECEPTION OF AUDIO AND/OR VIDEO MATERIAL**

(75) Inventors: **Anthony R Leaning**, Ipswich (GB); **Richard J Whiting**, Ipswich (GB)

(73) Assignee: **British Telecommunications public limited company**, London (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 908 days.

(21) Appl. No.: **10/433,259**

(22) PCT Filed: **Dec. 14, 2001**

(86) PCT No.: **PCT/GB01/05543**

§ 371 (c)(1),
(2), (4) Date: **Jun. 2, 2003**

(87) PCT Pub. No.: **WO02/49343**

PCT Pub. Date: **Jun. 20, 2002**

(65) **Prior Publication Data**

US 2004/0064573 A1 Apr. 1, 2004

(30) **Foreign Application Priority Data**

Dec. 15, 2000 (GB) 0030706.6
Mar. 30, 2001 (GB) 0108094.4
Nov. 19, 2001 (GB) PCT/GB01/05090

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

(52) **U.S. Cl.** **709/231**; 709/232; 709/233

(58) **Field of Classification Search** 709/230–237, 709/203, 217–219
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,608,450 A * 3/1997 Agarwal et al. 348/14.15
5,610,841 A * 3/1997 Tanaka et al. 725/115

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0669 587 8/1995

(Continued)

OTHER PUBLICATIONS

Avaro et al, "The MPEG-4 Systems and Description Languages: A Way Ahead in Audio Visual Information Representation", Signal Processing, Image Communication, NL, Elsevier Science Publishers, Amsterdam, vol. 9, No. 4, May 1, 1997, pp. 385-431, XP004075337.

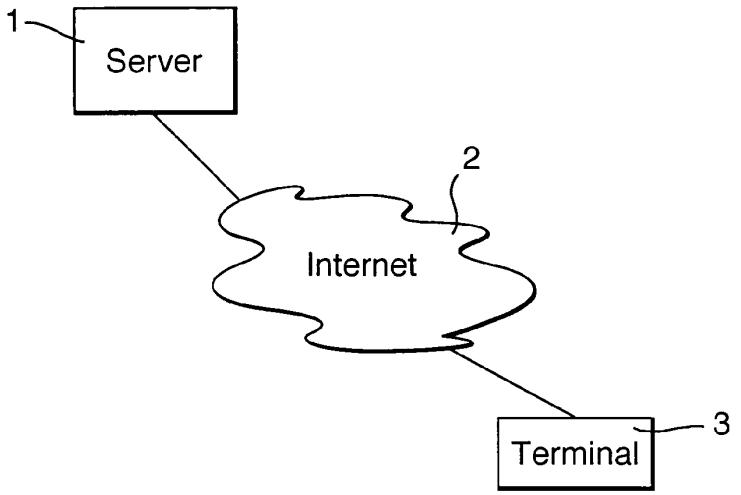
(Continued)

Primary Examiner—Thuha T. Nguyen
Assistant Examiner—Brendan Y Higa
(74) *Attorney, Agent, or Firm*—Nixon & Vanderhye, PC

(57) **ABSTRACT**

Delivery of recorded audio or video material over a telecommunications link from a server is accomplished by dividing the material into a sequence of sub-files each of which is independently requested by the terminal, which thereby has control of the rate of delivery. Provision may be made for switching between alternative sub-file sets representing alternative delivery modes or data rates.

38 Claims, 3 Drawing Sheets



U.S. PATENT DOCUMENTS

5,701,582	A	*	12/1997	DeBey	725/103
5,732,216	A		3/1998	Logan et al.	
5,835,495	A	*	11/1998	Ferriere	370/465
5,956,716	A		9/1999	Kenner et al.	
6,002,440	A	*	12/1999	Dalby et al.	375/240.12
6,016,166	A	*	1/2000	Huang et al.	348/515
6,032,189	A		2/2000	Jinzenji et al.	
6,118,790	A	*	9/2000	Bolosky et al.	370/468
6,452,943	B1		9/2002	Furuya	
6,760,808	B2	*	7/2004	Peters et al.	711/114
6,857,130	B2	*	2/2005	Srikantan et al.	725/93
6,993,788	B1	*	1/2006	Lawrence et al.	725/102
7,103,668	B1	*	9/2006	Corley et al.	709/231
2002/0062391	A1	*	5/2002	Densmore	709/245

FOREIGN PATENT DOCUMENTS

EP	0669587	A		8/1995
EP	669587	A2	*	8/1995
EP	0812112	A2		12/1997
EP	0924934	A		6/1999
WO	WO 97/22201			6/1997
WO	WO 99/22563			5/1999
WO	WO 99/34291			7/1999
WO	WO 99/52228			10/1999
WO	WO 00/42773			7/2000

OTHER PUBLICATIONS

Ramjee et al, "Adaptive Playout Mechanisms For Packetized Audio Applications in Wide-Area Networks", IEEE Comp. Soc. Press, US, Jun. 12, 1994, pp. 680-688, XP000496524.

"International Standard ISO/IEC 14772-1; 1997 Virtual Reality Modeling Language (VRML97)" 'Online!, 1997, pp. 1-236, XP002133320, URL:www.vrml.org/specifcations/VRML97.

* cited by examiner

Fig.1.

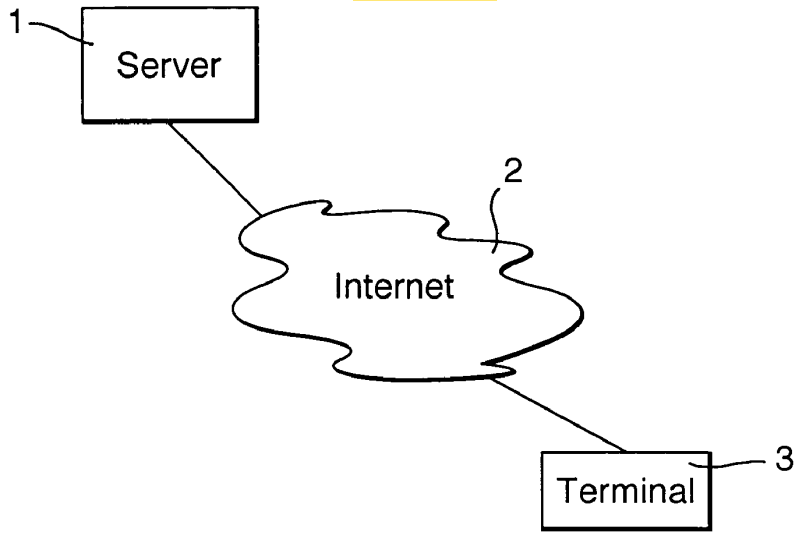

1
Server
2
Internet
Terminal
3

Fig.2.

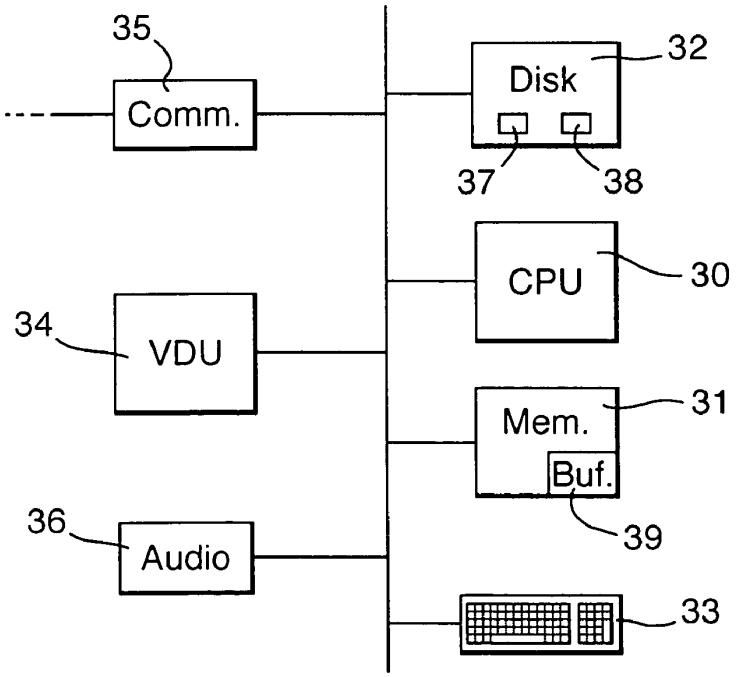

35
Comm.
Disk
32
37
38
CPU
30
34
VDU
Mem.
31
Buf.
39
36
Audio
33

Fig.3.

```
<html>
<head>

<!-- Odbits: Mode = “recorded” -->
<!--   Audio = “mp3-024k_11_s mp3-032k_11_s mp3-018k_11_s mp3-016k_11_m
mp3-008k_11_m” -->
<!-- LFI = “44” -->
<!-- SL = “178” -->
<!-- Title = “Toccata and Fugue in D minor” -->
<!-- Author = “J.S. Bach” -->
<!-- Created = “6 Nov 2000” -->
<!-- Copyright = “© 1999” -->
<!-- Comments = “One of Bach’s many organ works.” -->
</head>
<body>
An announcement to appear on the display
</body>
</html>
```

Fig.4.

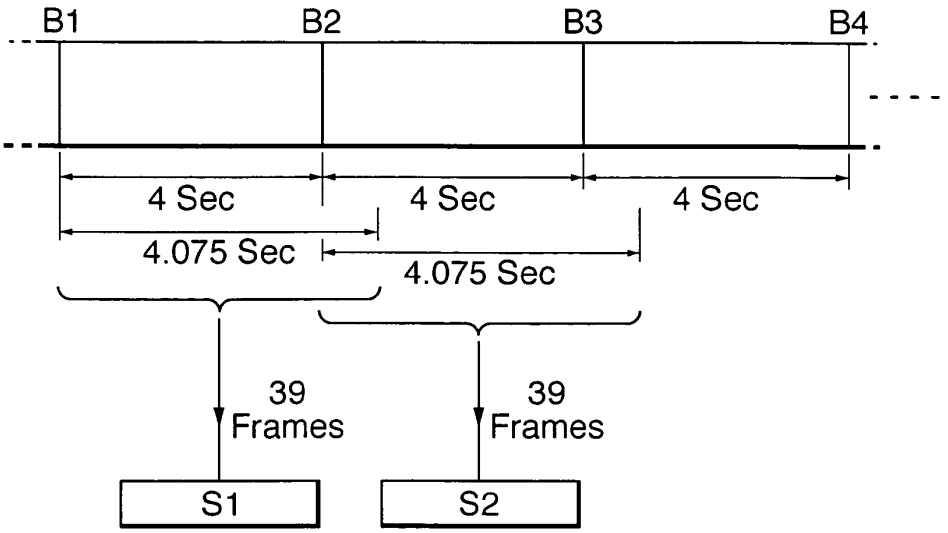
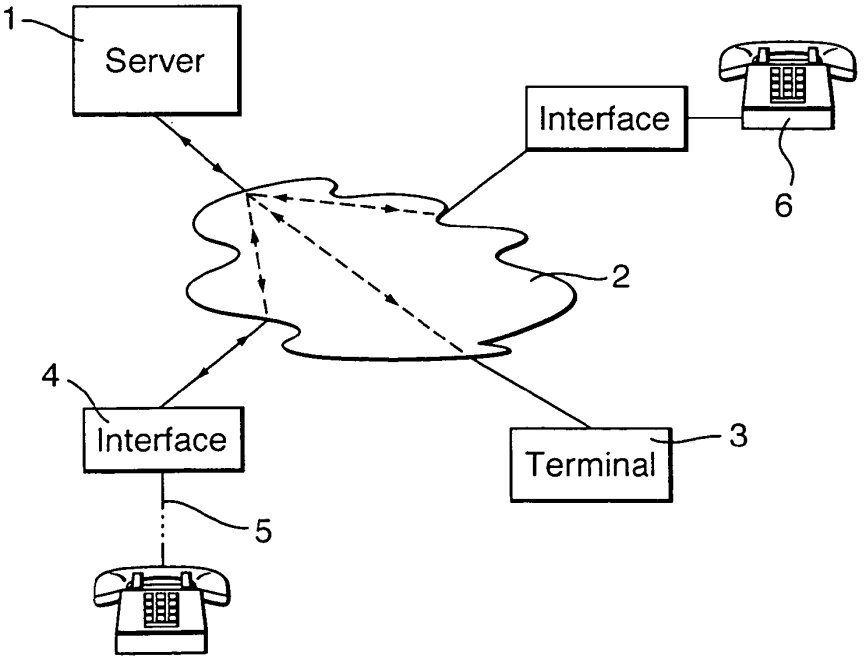

B1
B2
B3
B4
4 Sec
4 Sec
4 Sec
4.075 Sec
4.075 Sec
39 Frames
39 Frames
S1
S2

Fig.5.


1
Server
Interface
6
2
4
Interface
Terminal
3
5

TRANSMISSION AND RECEPTION OF AUDIO AND/OR VIDEO MATERIAL

This application is the US national phase of international application PCT/GB01/05543 filed 14 Dec. 2001 which designated the U.S.

BACKGROUND

1. Technical Field

The present invention is concerned with the delivery, over a telecommunications link, of digitally coded material for presentation to a user.

2. Related Art

In known systems of this type, a special server—often called a "streamer", controls delivery of material to a user terminal. Often, in the server, an item of material to be transmitted is stored as a single file; though U.S. Pat. No. 5,610,841 describes a video server which stores the material segmented into "media segment files". Another such system is described in published European patent application EP-A-669 587, where network congestion is accommodated by the terminal monitoring the contents of its receive buffer and, where appropriate, requesting the server to adjust its video data rate.

BRIEF SUMMARY

According to one aspect of the invention there is provided a terminal for playing audio or video material which is stored on a remote server as a set of files representing successive temporal portions of the said material, the terminal comprising:

a telecommunications interface for communication with the server;

a buffer for receiving the files from the telecommunications interface;

means for playing the contents of the buffer; and

control means responsive to the state of the buffer to generate request messages for further files for replenishment of the buffer.

In another aspect, the invention provides a method of transmitting digitally coded audio or video material comprising:

partitioning the material into a plurality of discrete files each representing successive temporal portions of the said material;

storing the files at a first station; and

at a second station—

a) transmitting to the first station requests for successive respective ones of the files;

b) receiving the files; and

c) decoding the files for replay of the material.

Other, optional, aspects of the invention are set out in the sub-claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Some embodiments of the present invention will now be described, with reference to the accompanying drawings, in which:

FIG. **1** is a diagram illustrating the overall architecture of the systems to be described;

FIG. **2** is a block diagram of a terminal for use in such a system;

FIG. **3** shows the contents of a typical index file;

FIG. **4** is a timing diagram illustrating a modified method of sub-file generation; and

FIG. **5** is a diagram illustrating a modified architecture.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

The system shown in FIG. **1** has as its object the delivery, to a user, of digitally coded audio signals (for example, of recorded music or speech) via a telecommunications network to a user terminal where the corresponding sounds are to be played to the user. However, as will be discussed in more detail below, the system may be used to convey video signals instead of, or in addition to, audio signals. In this example, the network is the internet or other packet network operating in accordance with the Hypertext Transfer Protocol (see RFCs 1945/2068 for details), though in principle other digital links or networks can be used. It is also assumed that the audio signals have been recorded in compressed form using the ISO MPEG-1 Layer III standard (the "MP3 standard"); however it is not essential to use this particular format. Nor, indeed, is it necessary that compression be used, though naturally it is highly desirable, especially if the available bit-rate is restricted or storage space is limited. In FIG. **1**, a server **1** is connected via the internet **2** to user terminals **3**, only one of which is shown. The function of the server **1** is to store data files, to receive from a user terminal a request for delivery of a desired data file and, in response to such a request, to transmit the file to the user terminal via the network. Usually such a request takes the form of a first part indicating the network delivery mechanism (e.g. http:// or file:// for the hypertext transfer protocol or file transfer protocol respectively) followed by the network address of the server (e.g. www.server **1**.com) suffixed with the name of the file that is being requested. Note that, in the examples given, such names are, for typographical reasons, shown with the "//" replaced by "\\".

In these examples, the use of the hypertext transfer protocol is assumed; this is not essential, but is beneficial in allowing use of the authentication and security features (such as the Secure Sockets Layer) provided by that protocol.

Conventionally, a server for delivery of MP3 files takes the form of a so-called streamer which includes processing arrangements for the dynamic control of the rate at which data are transmitted depending on the replay requirements at the user terminal, for the masking of errors due to packet loss and, if user interaction is allowed, the control of the flow of data between server and client; here however the server **1** need contain no such provision. Thus it is merely an ordinary "web server".

The manner in which the data files are stored on the server **1** will now be explained. Suppose that an MP3-format file has been created and is to be stored on the server. Suppose that it is a recording of J. S. Bach's Toccata and Fugue in D minor (BWV565) which typically has a playing time of 9 minutes. Originally this would have been created as a single data file, and on a conventional streamer would be stored as this one single file. Here, however, the file is divided into smaller files before being stored on the server **1**. We prefer that each of these smaller files is of a size corresponding to a fixed playing time, perhaps four seconds. With a compressed format such as MP3 this may mean that the files will be of different sizes in terms of the number of bits they actually contain. Thus the Bach file of 9 minutes duration would be divided into 135 smaller files each representing four seconds' playing time. In this example these are given file names which include a serial number indicative of their sequence in the original file, for example:

000000.bin

000001.bin

000002.bin

000003.bin

⋮

000134.bin

The partitioning of the file into these smaller sub-files may typically be performed by the person preparing the file for loading onto the web server **1**. (The expression "sub-files" is used here to distinguish them from the original file containing the whole recording: it should however be emphasised that, as far as the server is concerned, each "sub-file" is just a file like any other file). The precise manner of their creation will be described more fully below. Once created, these sub-files are uploaded onto the server in a conventional manner just like any other file being loaded onto a web server. Of course the filename could also contain characters identifying the particular recording (the sub-file could also be "tagged" with additional information—when you play an MP3 file you get information on the author, copyright etc), but in this example the sub-files are stored on the server in a directory or folder specific to the particular recording—e.g. mp3_bwv565. Thus a sub-file, when required, may be requested in the form:

http:\www.server**1**.com/mp3_bwv565/000003.bin

where "wvw.server**1**.com" is the URL of the server **1**.

It is also convenient for the person preparing the sub-files for loading onto the server to create, for each recording, a link page (typically in html format) which is also stored on the server (perhaps with filename mp3_bwv565/link.htm), the structure and purpose of which will be described later.

It is also convenient that the web server stores one or more (html) menu pages (e.g. menu.htm) containing a list of recordings available, with hyperlinks to the corresponding link pages.

Turning now to the terminal, this may typically take the form of a conventional desktop computer, with, however, additional software for handling the reception of the audio files discussed. If desired, the terminal could take the form of a handheld computer, or even be incorporated into a mobile telephone. Thus FIG. **2** shows such a terminal with a central processor **30**, memory **31**, a disk store **32**, a keyboard **33**, video display **34**, communications interface **35**, and audio interface ("sound card") **36**. For video delivery, a video card would be fitted in place of, or in addition to, the card **36**. In the disk store are programs which may be retrieved into the memory **31** for execution by the processor **30**, in the usual manner. These programs include a communications program **37** for call-up and display of html pages—that is, a "web browser" program such as Netscape Navigator or Microsoft Explorer, and a further program **38** which will be referred to here as "the player program" which provides the functionality necessary for the playing of audio files in accordance with this embodiment of the invention. Also shown is a region **39** of the memory **31** which is allocated as a buffer. This is a decoded audio buffer containing data waiting to be played (typically the playout time of the buffer might be 10 seconds). The audio interface or sound card **36** can be a conventional card and simply serves to receive PCM audio and convert it into an analogue audio signal, e.g. for playing through a loudspeaker. Firstly, we will give a brief overview of the operation of the terminal for the retrieval and playing of the desired recording when using the HTTP and an embedded or "plugin" client

1. The user uses the browser to retrieve and display the menu page menu.htm from the server **1**.
2. The user selects one of the hyperlinks within the menu page which causes the browser to retrieve from the server, and display, the link page for the desired recording—in this example the file mp3_bwv565_link.htm. The actual display of this page is unimportant (except that it may perhaps contain a message to reassure the user that the system is working correctly). What is important about this page is that it contains a command (or "embed tag") to invoke in the processor **30** a secondary process in which the player program **37** is executed. The invocation of a secondary process in this manner is well-known practice (such a process is known in Netscape systems as a "plug-in" and in Microsoft systems as "ActiveX"). Such commands can also contains parameters to be passed to the secondary process and in the system of FIG. **1** the command contains the server URL of the recording, which, for the Bach piece, would be http:\www.server**1**.com/mp3_bwv565.
3. The player program **37** includes an MP3 decoder, the operation of which is, in itself, conventional. Of more interest in the present context are the control functions of the program which are as follows.
4. The player program, having received the URL, adds to this the filename of the first sub-file, to produce a complete address for the sub-file—i.e. www.server**1**.com/mp3_bwv565/000000.bin. It will be observed that this system is organised on the basis that the sub-files are named in the manner indicated above, so that the terminal does not need to be informed of the filenames. The program constructs a request message for the file having this URL and transmits it to the server **1** via the communications interface **35** and the internet **2**. (Processes for translating the URL into an IP address and for error reporting of invalid, incomplete or unavailable URLs are conventional and will not therefore be described). We envisage that the player program would send the requests directly to the communications interface, rather than via the browser. The server responds by transmitting the required sub-file.
5. The player program determines from the file the audio encoding used in this sub-file and decodes the file back to raw PCM values in accordance with the relevant standard (MP3 in this example), making a note of the play time of this sub-file. Generally an audio file contains an identifier at the beginning of the file which states the encoding used. The decoded audio data is then stored in the audio buffer **38**.
6. The player program has a parameter called the playout time T_p. In this example it is set at **10** seconds (it could be made user-selectable, if desired). It determines the degree of buffering that the terminal performs.
7. The player program increments the filename to 000001.bin and requests, receives, decodes and stores this second sub-file as described in (4) and (5) above. It repeats this process until the contents of the buffer reach or exceed the playout time T_p. Note that it is not actually essential that the decoding occurs before the buffer but it simplifies matters as since the audio is decoded back to raw PCM then the duration of the buffered material is then explicitly known. It simplifies the control of the audio buffer if each of the sub-files is the same audio playback size.

8. Having reached the playout threshold T_p the decoded data are sent from the buffer to the audio interface **36** which plays the sound through a loudspeaker (not shown).
9. Whilst playing the sounds as in (8) above, the player program continually monitors the state of buffer fullness and whenever this falls below T_p it increments the filename again and obtains a further sub-file from the server. This process is repeated until a "file not found error" is returned.
10. If, during this process, the buffer becomes empty, the player program simply ceases playing until further data arrives.

The sub-file naming convention used here, of a simple fixed length sequence of numbers starting with zero, is preferred as it is simple to implement, but any naming convention can be used provided the player program either contains (or is sent) the name of the first sub-file and an algorithm enabling it to calculate succeeding ones, or alternatively is sent a list of the filenames.

It will have been observed that the system described above offers the user no opportunity to intervene in the replay process. Nor does it offer any remedy for the possibility of buffer underflow (due for example to network congestion). Therefore a second, more sophisticated embodiment of the invention, now to be described, offers the following further features:

a) the server stores two or more versions of the recording, recorded at different compression rates (for example at compressions corresponding to (continuous) data rates of 8, 16, 24 and 32 kbit/s respectively) and the player program is able to switch automatically between them.
b) the player program displays to the user a control panel whereby the user may start the playing, pause it, restart it (from the beginning, or from the point at which it paused), or jump to a different point in the recording (back or forward).

Note that these features are not interdependent, in that user control could be provided without rate-switching, or vice versa.

In order to provide for rate switching, the person preparing the file for loading onto the server prepares several source files—by encoding the same PCM file several times at different rates. He then partitions each source file into sub-files, as before. These can be loaded onto the server in separate directories corresponding to the different rate, as in the following example structure, where "008 k", "024 k" in the directory name indicates a rate of 8 kbit/s or 24 kbit/s and so on.

He also creates an index file (e.g. index.htm) the primary purpose of which is to provide a list of the data rates that are available.

Directory	Subdirectory	Filename
mp3__bwv565	none	link.htm index.htm
mp3__bwv565	008k__11__m	000000.bin 000001.bin 000002.bin 000003.bin . . . 000134.bin
mp3__bwv565	016k__11__m	000000.bin 000001.bin 000002.bin

-continued

Directory	Subdirectory	Filename
		000003.bin . . . 000134.bin
mp3__bwv565	018k__11__s	000000.bin 000001.bin 000002.bin 000003.bin . . . 000134.bin
mp3__bwv565	024k__11__s	000000.bin 000001.bin 000002.bin 000003.bin . . . 000134.bin
mp3__bwv565	032k__11__s	000000.bin 000001.bin 000002.bin 000003.bin . . . 000134.bin

Note that because the length of a sub-file corresponds, as explained earlier, to a fixed length of time, the number of sub-files is the same for each directory. The subdirectory names comprise the data rate in kbit/s (three digits) plus the letter "k"; in this example indications of the audio sampling rate (11.025 kHz) and a mono-stereo flag are appended, for verification purposes.

The index file would thus contain a statement of the form:

<!--Audio="024k__11_s 032k__11_s 018k_s 016k__11_m 008__11_m"-->

(The <!-- . . . --> simply indicates that the statement is embedded as a comment in an html file (or a simple text file could be used)). A typical index file is shown in FIG. **3** where other information is included: LFI is the highest sub-file number (i.e. there are 45 sub-files) and SL is the total playing time (178 seconds). "Mode" indicates "recorded" (as here) or "live" (to be discussed below). The other entries are either self-explanatory, or standard html commands.

Initially the player program will begin by requesting, from the directory specified in the link file, the index file, and stores locally a list of available data rates for future reference. (It may explicitly request this file or just specify the directory: most servers default to index.htm if a filename is not specified.) It then begins to request the audio sub-files as described earlier, from the first-mentioned "rate" directory in the index file—viz. 024k__11_s (or the terminal could override this by modifying this to a default rate set locally for that terminal). The process from then on is that the player program measures the actual data rate being received from the server, averaged over a period of time (for example 30 seconds). It does this by timing every URL request; the transfer rate achieved (number of bits per second) between the client and server is determined. The accuracy of this figure improves as the number of requests goes up. The player maintains two stored parameters which indicate, respectively, the current rate, and the measured rate.

The initiation of a rate change is triggered:

a) if the buffer ever empties AND the measured rate is less than the current rate AND the measured Buffer Low Percentage exceeds a Step Down Threshold (as described below), reduce the current rate; (changing at a time when the buffer is already empty is advantageous as the sound card is not playing anything and it may be necessary to reconfigure it if the audio sampling rate, stereo-mono setting or bit width (number of bits per sample) has changed).

b) if the measured rate exceeds not only the current rate but also the next higher rate for a given period of time (e.g. 120 seconds: this could if desired be made adjustable by the user) increase the current rate

The Buffer Low Percentage is the percentage of the time that the buffer contents represent less than 25% of the playout time (i.e. the buffer is getting close to being empty). If the Step Down Threshold is set to 0% then when the buffer empties the system always steps down when the other conditions are satisfied. Setting the Step Down Threshold to 5% (this is our preferred default value) means that if the buffer empties but the measured Buffer Low Percentage is greater than 5% it will not step down. Further buffer empties will obviously cause this measured rate to increase and will eventually empty the buffer again with a Buffer Low Percentage value exceeding 5% if the rate can not be sustained. Setting the value to 100% means the client will never step down.

The actual rate change is effected simply by the player program changing the relevant part of the sub-file address for example, changing "008k" to "024k" to increase the data rate from 8 to 24 kbit/s, and changing the current rate parameter to match. As a result, the next request to the server becomes a request for the higher (or lower) rate, and the sub-file from the new directory is received, decoded and entered into the buffer. The process just described is summarised in the following flowchart:

User	Terminal	Server
Select Menu page		
	Request http:\\server1.com/menu.htm	
		Send http:\\ server1.com/ menu.htm
	Display menu.htm	
Select item from Menu (Bach)		
	Extract hyperlink URL from menu.htm (mp3__bwv565/link.htm) Request http:\\server1.com/mp3__bwv565/link.htm	
		Send http:\\ server1.com/ mp3__ bwv565/ link.htm
	Display link.htm Execute secondary process (player program) specified in link.htm with parameters specified in link.htm (http:\\server1/ mp3__bwv565) Set Stem to that specified Set URL = Stem + "index.htm" Request this URL	
		Send requested file
	Set Rate List to rates specified in index.htm	

-continued

User	Terminal	Server
	Set LFI to value specified in index.htm Set StemC = Stem + "/" + Rate List(item 1) Set CurrentRate = rate specified in RateList (item 1) Set RateU = next higher rate in list or zero if none Set StemU = Stem + "/" + item in rate list corresponding to this rate; Set RateD = next lower rate in list or zero if none Set StemD = Stem + "/" + item in rate list corresponding to this rate; Set Current Subfile = 000000.bin J1: Set URL = StemC + Current Subfile Request this URL	
		If requested subfile exists, Send requested subfile; otherwise send error message
	If error message received, Stop Decode received subfile Write to buffer J1A: If Buffer Fullness > Tp seconds go to Step J3 J2: Increment Current Subfile Go to Step J1 J3: Begin/continue playing of buffer contents via sound card J4: If Buffer Fullness < Tp seconds go to Step J2 If BufferFullness = 0 AND Mrate < CurrentRate AND BufferLow % > Td go to Stepdown If MRate > NextRate AND NextRate <> 0 goto Stepup	
Input of user commands	If UserCommand = Pause then: Stop reading from buffer; Loop until Usercommand = Resume; go to J3 If UserCommand = Jump(j%)then: Clear buffer; Set CurrentSubfile = Integer[(LFI+1)*j/100]; go to Step J1 Go to Step J1A Stepup: Clear Buffer Set RateD = RateC Set StemD = StemC Set RateC = RateU Set StemC = StemU Set RateU = next higher rate in list or zero if none Set StemU = Stem + "/" + item in rate list corresponding to this rate Go to Step J1A Stepdown: Clear Buffer Set RateU = RateC Set StemU = StemC Set RateC = RateD Set StemC = StemD Set RateD = next lowerr rate in list or zero if none Set StemD = Stem + "/" + item in rate list corresponding to this rate Go to Step J1A	

The user control is implemented by the user being offered on the screen the following options which he can select using the keyboard or other input device such as a mouse:

a) Start: implement the numbered steps given above, from step **4**. Whether, when a recording is first selected, it begins to play automatically, or requires a Start instruction from the user, is optional; indeed, if desired, the choice may be made by means of an additional "autoplay" parameter in the link file.

b) Pause: implemented by an instruction to the MP3 decoder to suspend reading data from the buffer;

c) Resume: implemented by an instruction to the MP3 decoder to resume reading data from the buffer;

d) Jump: implemented by the user indicating which part of the recording he wishes to jump to—for example by moving a cursor to a desired point on a displayed bar representing the total duration of the recording; the player then determines that this point is x % along the bar and calculates the number of the next sub-file needed, which is then used for the next request. In the Bach example with 125 sub-files then a request to play from a point 20% into the recording would result in a request for the 26th sub-file—i.e. 000025.bin. It will be apparent that this calculation is considerably simplified if each sub-file corresponds to the same fixed duration. We prefer, in the case of the jump, to suspend decoding and clear the buffer so that the new request is sent immediately, but this is not actually essential.

Alternatively, the user may be presented with a list of text labels or indices which may be selected (e.g. by means of a mouse) to initiate a jump. This might be implemented as follows:

The index.htm file held by the terminal, in memory, contains lines of the form (assuming that this information is embedded as comments within the document.

<--!Odbits: Index="0:01:44 Wireless"-->.

where where "Odbits:" is a keyword indicating to the player program that the following text is to be processed by a program, "Index" indicates the function that the player program is to perform, and the text with the quotation marks consists of the time (hours:minutes:seconds) from the beginning of the recording at which playing is to commence.

The player program reads the index.htm file and if it recognises an index command it generates a message ("an event") to the link.htm page which contains commands (typically written in Javascript) to handle such an event by displaying the labels at a desired place on the displayed page and respond to selection of those commands by generating a jump (containing the corresponding time) to the player program.

It is of interest to discuss further the process of partitioning the original file into sub-files. First, it should be noted that if (as in the first version described above), there is no expectation that a sub-file will be followed by a sub-file other than that which immediately follows it in the original sequence, then it matters little where the boundaries between the sub-files are located. In that case the sub-file size can be a fixed number of bits, or a fixed playing time length (or neither of these)—the only real decision is how big the sub-files should be. Where jumps are envisaged (in time, or between different data rates) there are other considerations. Where, as with many types of speech or audio coding (including MP3), the signal is coded in frames, a sub-file should contain a whole number of frames. In the case of rate switching, it is, if not actually essential, highly desirable that the sub-file boundaries are the same for each rate, so that the first sub-file received for a new rate continues from the same point in the recording that the last sub-file at the old rate ended. To arrange that every sub-file should represent the same fixed time period (e.g. the 4 seconds mentioned above) is not the only way of achieving this, but it is certainly the most convenient. Note however that, depending on the coding system in use, the requirement that a sub-file should contain a whole number of frames may mean that the playing duration of the sub-files does vary slightly. Note that in this embodiment of the invention, the available data rates, though they use different degrees of quantisation, and differ as to whether they encode in mono or stereo, all use the same audio sampling rate and in consequence the same frame size. Issues that need to be addressed when differing frame sizes are used are discussed below.

As for the actual sub-file length, excessively short sub-files should preferably be avoided because (a) they create extra network traffic in the form of more requests, and (b) on certain types of packet networks—including IP networks—they are wasteful in that they have to be conveyed by smaller packets so that overhead represented by the requesting process and the packet header is proportionately greater. On the other hand, excessively large sub-files are disadvantageous in requiring a larger buffer and in causing extra delay when starting play and/or when jumps or rate changes are invoked. A sub-file size of between 30% and 130% of the playout time, or preferably around half the playout time (as in the examples given above), is found to be satisfactory.

The actual process of converting the sub-files can be implemented by means of a computer programmed in accordance with the criteria discussed. Probably it will be convenient to do this on a separate computer, from which the sub-files can be uploaded to the server.

Another refinement that can be added is to substitute a more complex sub-file naming convention so as to increase security by making it more difficult for an unauthorised person to copy the sub-files and offer them on another server. One example is to generate the filenames using a pseudo-random sequence generator, e.g. producing filenames of the form:

0130254613464367753454313 4.bin
9454345234543453345213 4565.bin
. . .

In this case the player program would include an identical pseudo-random sequence generator. The server sends the first filename, or a "seed" of perhaps four digits, and the generator in the player can then synchronise its generator and generate the required sub-file names in the correct sequence.

In the above example of rate-switching, all the data rates used had the same frame size, specifically they used MP3 coding of PCM audio sampled at 11.025 KHz and a (PCM) frame size of 1152 samples. If it is desired to accomplish rate switching between MP3 (or other) recordings having different frame sizes, problems arise due to the requirement that a sub-file should contain a whole number of frames, because the frame boundaries do not then coincide. This problem can be solved by the following modified procedure for creating the sub-files. It should be noted particularly that this procedure can be used in any situation where rate switching is required and is not limited to the particular method of delivery discussed above.

FIG. **4** shows diagrammatically a sequence of audio samples, upon which successive four-second segments are delineated by boundary marks (in the figure) B**1**, B**2** etc. At 11.025 KHz, there are 44,100 samples in each segment.

1. Encode the audio, starting at boundary B**1**, frame by frame, to create an MP3 sub-file, continuing until a whole number of frames having a total duration of at least four seconds

has been encoded. With a frame size of 1152 samples, four seconds corresponds to 38.3 frames, so a sub-file S**1** representing 39 frames will actually be encoded, representing a total duration of 4.075 seconds.

2. Encode the audio, in the same manner, starting at boundary B**2**.
3. Repeat, starting each time at a 4-second boundary, so that in this way a set of overlapping sub-files is generated for the whole audio sequence to be coded. The last segment (which may well be shorter than four seconds) has of course nothing following it, and is padded with zeroes (i.e. silence).

Coding of the other data rates using different frame sizes proceeds in the same manner.

At the terminal, the control mechanisms are unchanged, but the decoding and buffering process is modified:

1. Receive sub-file S**1**;
2. Decode sub-file S**1**;
3. Write into the buffer only the first four seconds of the decoded audio samples (discard the remainder);
4. Receive sub-file S**2**;
5. Decode sub-file S**2**;
6. Write into the buffer only the first four seconds of the decoded audio samples;
7. Continue with sub-file S**3** etc.

In this way, it is ensured that the sub-file sets for all rates have sub-file boundaries which correspond at the same points in the original PCM sample sequence.

Thus, each four-second period except the last is, prior to encoding, "padded" with audio samples from the next four-second period so as to bring the sub-file size up to a whole number of MP3 frames. If desired, the padding samples could be taken from the end of the preceding four-second period instead of (or as well as) the beginning of the following one.

Note that the MP3 standard allows (by a scheme known as "bit reservoir") certain information to be carried over from one audio frame to another. In the present context, while this is acceptable within a sub-file, it is not acceptable between sub-files. However, since naturally the standard does not allow such carry-over at the end or beginning of a recording, this problem is easily solved by encoding each sub-file separately, as if it were a single recording.

Changes of sampling rate (and indeed switching between mono and stereo operation) have some practical implications for operation of the audio interface **36**. Many conventional sound cards, although capable of operation at a range of different settings, require re-setting in order to change sampling rate, and necessarily this causes an interruption in its audio output. Thus in a further modification, we propose that the sound card could be run continuously at the highest sampling rate envisaged. When the player program is supplying, to the buffer, data at a lower sampling rate, this data is then up-sampled to this highest rate before or after the buffer. Similarly, if the card is always operated in stereo mode, decoded mono signals can be fed in parallel so feed both the left and right channels of the sound card input. Again, if the number of bits per sample of the decoded signal is lower than expected by the card, the number of bits can be increased by padding with zeros.

Recollecting that the criteria discussed earlier for automatic data rate switching downwards envisaged a rate reduction only in cases of buffer underflow (involving therefore interruptions in the output), we note that with this modification such interruption can be avoided and therefore it is preferable to employ a criterion which anticipates underflow and avoids it in the majority of cases. In this case the first of the three AND conditions mentioned above (namely, that the buffer is empty) would be omitted.

Another feature that may be provided in the context of the system described is the display of visual information such as subtitles, or still pictures (in the nature of a slide show) to accompany the sound.

This might be implemented as follows:

(a) the document (link.htm)y being displayed contains formatting information specifying the location in the page at which the visual information is to appear;

(b) the file index.htm contains lines of the form (assuming that this information is embedded as comments with the document.

<!--Odbits: Subtitle="0:1:01 Subtitle text"-->

<!--Odbits: Image="0:2:04 http:// . . . /picture.jpg"-->

where "Odbits:" is a keyword indicating to the player program that the following text is to be processed by a program, "Subtitle" or "Image" indicates the function that the player program is to perform, and the text with the quotation marks consists of the time (hours:minutes:seconds) from the beginning of the recording at which the corresponding display is to commence, followed by the actual subtitle or the URL of the desired image, as the case may be.

(c) the player program computes the actual time, being the product of the current subfile index and the subfile length, incremented by the time the current subfile has been playing.

(d) the player program compares the actual time with the times embedded in the index.htm file and in the event of a match returns a message to the link.htm page (which invoked the player originally). Such a message is referred to as an "event". How the link.htm page handles the event is at the discretion of the person writing the link.htm page. Typically

(e) for a subtitle, the link.htm page contains executable commands (typically written in Javascript) which respond to the event by reading the subtitle text from the index.htm file and displaying it at the desired place.

(f) for an image, the link.htm page contains such commands to respond to the event by reading the URL of the image from the index.htm file, generating a request message for downloading the image file having that URL and displaying it at the desired place on the page. This downloading could occur at the time that the image is required: alternatively the user could be offered the option of having the image files pre-loaded. If he accepts this option then the player program would, prior to any audio replay, download all the image files listed and store them in a local cache.

The same principle may be applied to the delivery of video recordings, or of course, video recordings with an accompanying sound track. In the simpler version, where there is only one recording, the system differs from the audio version only in that the file is a video file (e.g. in H.261 or MPEG format) and the player program incorporates a video decoder. The manner of partitioning the file into sub-files is unchanged.

As in the audio case, there may be two or more recordings corresponding to different data rates, selected by the control mechanism already described. Also one can provide additional recordings corresponding to different replay modes such as fast forward or fast reverse which can be selected by an extension of the user control facilities already described. Again, a systematic convention for file and directory naming can be followed so that the player program can respond to—for example—a fast forward command by amending the sub-file address.

The delivery of video recordings does however have further implications for file partitioning if switching or jumps are to be permitted. In the case of recordings where each frame of

a picture is coded independently, it is sufficient that a sub-file contains a whole number of frames of a picture. If compression involving inter-frame techniques is in use, however, the situation is more complex. Some such systems (for example the MPEG standards) generate a mixture of independently coded frames ("intra-frames") and predictively coded frames; in this case each sub-file should preferably begin with an intra-frame.

In the case of inter-frame coding systems such as the ITU H.261 standard, which do not provide for the frequent, regular inclusion of intra-frames, this is not possible. This is because—taking rate-switching as an example, if one were to request sub-file n of a higher bit rate recording followed by sub-file n+1 of a lower bit-rate recording, the first frame of the lower bit-rate sub-file would have been coded on an inter-frame basis using the last decoded frame of sub-file n of the lower rate recording, which of course the terminal does not have at its disposal—it has the last decoded frame of sub-file n of the higher rate recording. Thus serious mistracking of the decoder would occur.

In the case of switching between normal play and a fast play mode, the situation is in practice slightly different. On fast forward play at, for example, 5 times normal speed, one encodes only every 5th frame. In consequence the inter-frame correlation is much reduced and inter-frame coding becomes unattractive, so one would generally prefer to encode a fast play sequence as intra-frames. Switching from normal to fast then presents no problem, as the intra-frames can be decoded without difficulty. However, when reverting to normal play, the mistracking problem again occurs because the terminal is then presented with a predictively coded frame for which it does not have the preceding frame.

In either case the problem can be solved by using the principle described in our international patent application No. WO98/26604 (issued in USA as U.S. Pat. No. 6,002,440). This involves the encoding of an intermediate sequence of frames which bridges the gap between the last frame of the preceding sequence and the first frame of the new sequence.

The operation of this will now be described in the context of fast forward operation (fast rewind being similar but in reverse). In this example we assume that a 9 minute video sequence has been encoded at 96 kbit/s according to the H.261 standard, and again at 5 times normal rate entirely at H.261 infra-frames, and that the resulting files have each been partitioned into four-second sub-files. Here, four seconds refers to the duration of the original video signal, not to the fast forward playing time. Following a naming convention similar to that employed above, the sub-files might be:

Directory	Subdirectory	Filename
mpg_name	096k_x1	000000.bin 000001.bin . . . 000134.bin
	096k_x5	000000.bin . . . 000134.bin

where "name" is a name to identify the particular recording, "x1" indicates normal rate and "x5" indicates five times normal rate—i.e. fast forward.

To switch from normal play to fast forward is only necessary for the player program to modify the sub-file address to point to the fast forward sequence—e.g.

Request mpg_name/096k_x1/000055.bin
is followed by
Request mpg_name/096K_x5/000056.bin

In order to construct the bridging sequences for switching back to normal play it is necessary to construct a bridging sub-file for each possible transition. As described in our international patent application mentioned above, a sequence of three or four frames is generally sufficient for bridging, so a simple method of implementation is to construct bridging sub-files of only 4 frames duration—e.g.

Directory	Subdirectory	Filename
mpg_name	096K_5 > 1	0000001.bin . . . 000133.bin

So that the switching is accomplished by a series of requests such as:

Request mpg_name/096k_x5/000099.bin
Request mpg_name/096k_5>1/000099.bin
Request mpg_name/096k_x1/000100.bin

The bridging sub-file is generated as follows:

Decode the fast forward sequence to obtain a decoded version of the last frame of sub-file 99, (at 25 frames per second this will be frame 100,000 of the original video signal).

Decode the normal sequence to obtain a decoded version of the first frame of sub-file 100 (i.e. frame 100,001). Re-encode this one frame four times using H.261 inter-frame coding based on the decoded frame 100,000 as the initial reference frame.

Thus, when the decoder has decoded the fast forward sub-file, followed by the bridging sub-file it will have reconstructed frame 100,000 correctly and will be ready to decode the normal (x1) frames. Incidentally, the reason that one encodes the same frame several times in this procedure is that doing so merely once, would produce poor picture quality due to the quantisation characteristics of H.261.

Exactly the same process could be used for rate-switching (albeit that now bridging sub-files are required in both directions). However, it will be observed that, as described, the bridging sub-file results in a freezing of the picture for a period of four frames—i.e. (at 25 frames per second) 160 ms. In switching from fast to normal play this is acceptable—indeed one would probably choose to clear the buffer at this point. It may or may not be subjectively acceptable on rate-switching. An alternative therefore would be to construct a four-second bridging sequence. The request series would then look like:

mpg_name/096k_x1/000099.bin
mpg_name/096/128_x1/000100.bin
mpg_name/128k_x1/000101.bin

The bridging sub-file would in that case be constructed either by recoding the fifth decoded frame of the decoded 128 kbit/s sequence four times starting with decoded 96 kbit/s frame 100,000 as the reference frame, or coding the first four frames of the decoded 128 kbit/s sequence starting with decoded 96 kbit/s frame 100,000 as the reference frame. In both cases the remaining 96 frames of the bridging sub-file would be a copy of the 128 kbit/s sub-file.

The files to be delivered have been referred to as "recordings". However, it is not necessary that the entire audio or video sequence should have been encoded—or even exist—

before delivery is commenced. Thus a computer could be provided to receive a live feed, to code it using the chosen coding scheme, and generate the sub-files "on the fly" and upload them to the server, so that, once a few sub-files are present on the server, delivery may commence.

One application of this delivery system would be for a voice-messaging system, as illustrated in FIG. **5** where the server **1**, network **2** and terminal **3** are again shown. A voice-messaging interface **4** serves to receive telephone calls, for example via the public switched telephone network (PSTN) **5**, to record a message, encode it, partition it into sub-files, and upload them to the server **1**, where they can be accessed in the manner described earlier. Alternatively a second interface **6** could be provided, operating in a similar manner to the terminal **3** but controlled remotely via the PSTN by a remote telephone **5**, to which the replayed audio signals are then sent.

The same system can be used for a live audio (or video) feed. It is in a sense still "recorded"—the difference being primarily that delivery and replay commence before recording has finished, although naturally there is an inherent delay in that one must wait until at least one sub-file has been recorded and loaded onto the server **1**.

The system can proceed as described above, and would be quite satisfactory except for the fact that replay would start at the beginning whereas what the user will most probably want is for it to start now—i.e. with the most recently created sub-file.

With a lengthy audio sequence one may choose to delete the older sub-files to save on storage: with a continuous feed (i.e. 24 hours a day) this will be inevitable and moreover one would need to reuse the sub-file names (in our prototype system we use 000000.bin to 009768.bin and then start again at 000000.bin), so that the older sub-files are constantly overwritten with the new ones. A simple method of ensuring delivery starting with the most recent sub-file would be to include in the index file an extra command instructing the player program to start by requesting the appropriate sub-file. This however has the disadvantage that the index file has to be modified very frequently—ideally every time a new sub-file is created. Therefore we propose a method whereby the player program scans the server to find the starting sub-file, as follows. In the index file, the Mode parameter is set to "live" to trigger the player program to invoke this method. LFI is set to indicate the maximum number of sub-files that may be stored—say 9768. The method involves the following steps and presupposes that (as is conventional) each sub-file's "last modified" time and date has been determined. When using the HTTP protocol this can be achieved using a HEAD request which results not in delivery of the requested sub-file but only of header information indicating the time that the sub-file was written to the server, or zero if the sub-file does not exist. This time is represented below as GetURL(LiveIndex) where LiveIndex is the sequence number of the sub-file in question. Comments are preceded by "//".

```
1   LFI = 9768 // read from the index.htm file
    LiveIndex = LFI / 2
    StepSize = LFI / 2
    LiveIndexModifiedAt = 0; // the beginning of time.
10  ThisIndexWasModifiedAt = GetURL(LiveIndex);
20  If (StepSize = 1)
    {
     // LiveIndexModifiedAt contains the time the file was written or 0
     if no file
     // has been found. LiveIndex contains the index.
     goto 30
```

-continued

```
    }
    StepSize = StepSize / 2
    if (ThisIndexWasModifiedAt > LiveIndexModifiedAt)
    {
     LiveIndexModifiedAt = ThisIndexWasModifiedAt;
     LiveIndex = LiveIndex + StepSize
    }
    else
    {
     LiveIndex = LiveIndex − StepSize
    }
    Goto 10
30  FINISH
```

Having found the LiveIndex it is prudent to step back the T_p (playout time) and start to make the requests to fill the audio buffer from there. Playing may commence in the normal way.

Once the recording has actually finished, the index file can if desired be modified to set Mode to "recorded", and any length parameters.

If desired the player program could check periodically to see whether the index file has changed from "live" to "recorded" mode and if so to switch to "recorded" mode playing.

A simpler and much faster method of the identification of the "latest" sub-file will now be described, assuming, first of all, a single continuous sub-file numbering sequence.

1. Terminal issues a HEAD request for the first sub-file (e.g. 000000.bin).

2. The server replies by sending the header of this file and includes the date and time the file was last modified (MODTIME) and the date and time at which this reply was sent (REPLYTIME) (both of these are standard http. fields).

3. The terminal calculates the elapsed time (ELTIME) by subtracting the two (ELTIME=REPLYTIME−MODTIME), and divides this by the playing duration of a sub-file (4 seconds, in these examples) to obtain LIVEINDEX=ELTIME/4.

4. The terminal calculates the filename of the sub-file having this index.

5. The terminal issues a HEAD request with this filename and if necessary each subsequent filename until it receives zero (file not found) whereupon it regards the latest sub-file which is found as the "Current sub-file".

6. The terminal begins requesting files, starting at point J**1**: of the flowchart given earlier.

This method is considerably faster than that described above for the cyclically numbered sub-files. Note that older sub-files may still be deleted, to reduce storage requirement, as long as the starting sub-file is kept. The method can however be modified to accommodate filename re-use (cyclic addresses), but would require:

(i) That the starting sub-file name (e.g. 000000.bin) is not re-used so that it is always available to supply the header information at Step 2. Thus, with wrapping at 009768.bin, sub-file 009768.bin would be followed by sub-file 000001.bin.

(ii) The calculated LIVEINDEX at Step 3 is taken Modulo 9768 (i.e. the remainder when ELTIME/4 is divided by 9768).

(iii) Sub-file deletion always leads the creation of new sub-files so that a few file-names between the newest sub-file and the oldest undeleted sub-file do not exist, in order that the expected "file not found" response occurs at Step 5.

There may be a danger of the playing operation running slightly faster or slower than the recording operation. To guard against the former it may be arranged that the player program checks each sub-file it receives to ascertain whether

it is marked with a later time than the previous one: if not the sub-file is discarded and repeated requests made (perhaps three times) followed by a check of the index file if these requests are unsuccessful.

If the playing lags behind the recording process this can be identified by the player program occasionally checking the server for the existence of a significant number of sub-files more recent than those currently being requested, and if such sub-files do exist, initiating a "catching up" process—e.g. by regularly discarding a small amount of data.

What is claimed is:

1. A terminal for playing audio or video material which is stored on a remote server as a set of files representing successive temporal positions of said material, the terminal comprising:

A telecommunications interface for communication with the server;

A buffer for receiving the files from the telecommunications interface;

Means for playing the contents of the buffer;

Control means to determine addresses of further files to be requested and in response to the state of the buffer to generate request messages, containing such addresses, fur further files for replenishment of the buffer; and

Means for identifying a starting file for decoding by an iterative process of trial requests comprising generating a trial request for a first file, receiving a rely indicating whether the requested file exists, if the requested file exists generating a trial request for a later file, or if it does not exist, generating a trial request for an earlier file.

2. A terminal according to claim **1** arranged in operation to decode the files before storing them in the buffer.

3. A terminal according to claim **1** in which the files are allocated addresses according to a predetermined algorithm and the terminal includes means operable for a said request message to calculate, in accordance with said algorithm, an address for inclusion in the request message.

4. A method of transmitting digitally coded audio or video material, said method comprising:

partitioning the material into a plurality of discrete files each representing successive temporal portions of the said material;

storing the files at a first station; and

at a second station

a) transmitting to the first station requests for successive respective ones of the files;

b) receiving the files;

c) decoding the files for replay of the material;

d) generating a trial request for a first file,

e) receiving from the first station a reply including data representing the original time of the first file and the time of said reply, and

f) estimating from these data an estimated identity of a most recent file at the first station.

5. A method according to claim **4** in which the material is stored in a buffer and the request messages are generated in dependence on the state of the buffer.

6. A method according to claim **5** in which the material is stored in the buffer after decoding.

7. A method according to claim **6** in which said material is audio material, in which:

a) the files are encoded using an audio coding method having a frame structure;

b) the step of partitioning the material into a plurality of discrete files comprises notionally dividing the material into a plurality of temporal portions and generating each said file, other than the last, by coding one respective temporal portion and an initial part of the following portion such that the portions together represent a whole number of frames;

c) after decoding, that part of the decoded material which corresponds to said initial part of the following portion is discarded and

wherein that part of the decoded material which corresponds to said initial part of the following portion is not stored in the buffer.

8. A method according to claim **4** of transmitting digitally coded audio or video material, said method comprising:

partitioning the material into a plurality of discrete files each representing successive temporal portions of the said material;

storing the files at a first station; and

at a second station

a) transmitting to the first station requests for successive respective ones of the files;

b) receiving the files; and

c) decoding the files for replay of the material;

including storing a plurality of sets of files, which sets correspond to respective different delivery modes, and including, at the second station, effecting mode switching by providing that subsequent request messages shall request files from a set different from the set to which the immediately preceding request related.

9. A method according to claim **8** in which at least some of said sets of files correspond to respective different data rates, including:

monitoring the received data rate at the second station; and

in the event that the measured rate is below that needed for the set to which the currently requested field belongs, performing mode switching to provide that subsequent said request messages shall request files from a set corresponding to a lower data rate.

10. A method according to claim **8** in which at least some of said sets of files correspond to respective different data rates, including:

monitoring the received data rate at the second station; and

in the event that the measured rate is sufficient to support delivery of files of a higher data rate than that of the set to which the currently requested file belongs, performing mode switching to provide that subsequent said request messages shall request files from a set corresponding to a higher data rate.

11. A method according to claim **8** in which at least some of said sets of files correspond to respective different playing modes, in which:

the second station includes means to receive commands from a user of the second station to perform said mode switching from a current playing mode to a desired playing mode corresponding to that command; and

the mode switching means is operable, upon receipt of the command, to provide that subsequent said request messages shall request files from a set corresponding to said desired playing mode.

12. A method according to claim **8** in which said material is in the form of video recordings, at least some of said files having been encoded using, for at least some frames thereof, inter-frame coding, and including, at the second station, before generating the request message for a file from a different set, generating a request message for a file for correction of decoder tracking.

13. A method according to claim **8** in which each said address has a first part and a second part, wherein all files of a given mode have the same first part and wherein files that are of different modes but correspond to the same temporal por-

tion of said audio or video material have the same second part, whereby mode switching may be effected by changing the first part of the address.

14. A method according to claim **4** in which said material is audio material, in which:

a) the files are encoded using an audio coding method having a frame structure;

b) the step of partitioning the material into a plurality of discrete files comprises notionally dividing the material into a plurality of temporal portions and generating each said file, other than the last, by coding one respective temporal portion and an initial part of the following portion such that the portions together represent a whole number of frames;

c) after decoding, that part of the decoded material which corresponds to said initial part of the following portion is discarded.

15. A method according to claim **14** in which said sets comprise a first set and a second set for which the frame length is different from that used for encoding the second set, and the division into temporal portions is the same for both sets.

16. A method according to claim **4** in which said material is audio material, in which:

a) the files are encoded using an audio coding method having a frame structure;

b) the step of partitioning the material into a plurality of discrete files comprises notionally dividing the material into a plurality of temporal portions and generating at least some of said files by coding one respective temporal portion and so much of the end of the immediately preceding temporal portion and/or the beginning of the immediately following temporal portion as to constitute with said one respective said temporal portion, a whole number of frames of said frame structure; and

c) after decoding, that part of the decoded material which corresponds to said end of the immediately preceding temporal portion and/or beginning of the immediately following temporal portion is discarded.

17. A method according to claim **4** in which the files are allocated addresses according to a predetermined algorithm, the method including calculating, at the second station in accordance with said algorithm, an address for inclusion in a request message.

18. A method according to claim **17** in which the algorithm generates the addresses in accordance with a key, including the step of transmitting the key to the second station for use in calculating the addresses.

19. A method according to claim **18** in which the algorithm generates the addresses in accordance with a pseudo-random sequence.

20. A method according to claim **19** in which the algorithm generates the addresses in accordance with a pseudo-random sequence and the key is a seed value for setting the start point of the pseudo-random sequence.

21. A method according to claim **4** including, at the second station:

(a) receiving from the first station a list containing times and data defining actions to be performed at those times;

(b) computing a time, relative to the beginning of the material, represented by the current playing point;

(c) comparing the computed time with the times in the list and, in the event of a match generating a command containing the respective data for initiation of the action.

22. A method according to claim **21** in which the actions include the display of a subtitle.

23. A method according to claim **21** in which the actions include the display of an image.

24. A method according to claim **23** including transmitting to the first station a request for images identified by the said data, and storing the images at the second station until required for display.

25. A method, of transmitting digitally coded audio or video material, said method comprising:

partitioning the material into a plurality of discrete files each representing successive temporal portions of the said material;

storing the files at a first station; and

at a second station

a) transmitting to the first station requests for successive respective ones of the files;

b) receiving the files; and

c) decoding the files for replay of the material;

d) wherein the second station identifies a starting file for decoding by an iterative process of trial requests comprising generating a trial request for a first file, receiving from the first station a reply indicating whether the requested file exists, if the requested file exists, generating a trial request for a later file or, if it does not exist, generating a trial request for an earlier file.

26. A method according to claim **25** in which the material is stored in a buffer and the request messages are generated in dependence on the state of the buffer.

27. A method according to claim **26** in which the material is stored in the buffer after decoding.

28. A method according to claim **27** in which said material is audio material, in which:

a) the files are encoded using an audio coding method having a frame structure;

b) the step of partitioning the material into a plurality of discrete files comprises notionally dividing the material into a plurality of temporal portions and generating each said file, other than the last, by coding one respective temporal portion and an initial part of the following portion such that the portions together represent a whole number of frames;

c) after decoding, that part of the decoded material which corresponds to said initial part of the following portion is discarded and

wherein that part of the decoded material which corresponds to said initial part of the following portion is not stored in the buffer.

29. A method according to claim **25** in which said material is audio material, in which:

a) the files are encoded using an audio coding method having a frame structure;

b) the step of partitioning the material into a plurality of discrete files comprises notionally dividing the material into a plurality of temporal portions and generating each said file, other than the last, by coding one respective temporal portion and an initial part of the following portion such that the portions together represent a whole number of frames;

c) after decoding, that part of the decoded material which corresponds to said initial part of the following portion is discarded.

30. A method according to claim **25** in which said material is audio material, in which:

a) the files are encoded using an audio coding method having a frame structure;

b) the step of partitioning the material into a plurality of discrete files comprises notionally dividing the material into a plurality of temporal portions and generating at

least some of said files by coding one respective temporal portion and so much of the end of the immediately preceding temporal portion and/or the beginning of the immediately following temporal portion as to constitute with said one respective said temporal portion, a whole number of frames of said frame structure; and

c) after decoding, that part of the decoded material which corresponds to said end of the immediately preceding temporal portion and/or beginning of the immediately following temporal portion is discarded.

31. A method according to claim **30** in which said sets comprise a first set and a second set for which the frame length is different from that used for encoding the second set, and the division into temporal portions is the same for both sets.

32. A method according to claim **25** in which the files are allocated addresses according to a predetermined algorithm, the method including calculating, at the second station in accordance with said algorithm, an address for inclusion in a request message.

33. A method according to claim **32** in which the algorithm generates the addresses in accordance with a key, including the step of transmitting the key to the second station for use in calculating the addresses.

34. A method according to claim **33** in which the algorithm generates the addresses in accordance with a pseudo-random sequence.

35. A method according to claim **34** in which the algorithm generates the addresses in accordance with a pseudo-random sequence and the key is a seed value for setting the start point of the pseudo-random sequence.

36. A terminal for playing audio or video material which is stored on a remote server as a set of files representing successive temporal positions of the said material, the terminal comprising:

A telecommunications interface for communication with the server;

a buffer for receiving the files from the telecommunications interface;

means for playing the contents of the buffer; and

control means to determine the addresses of further files to be requested and to determine the addresses of further files to be requested and in response to the state of the buffer to generate request messages, containing such addresses, for further files for replenishment of the buffer, wherein said control means further generating a trial request for a first file, receiving a reply including data representing the original time of the first file and the time of said reply;

and estimating from these data an estimated identity of the most recent file.

37. A terminal according to claim **36** arranged in operation to decode the files before storing them in the buffer.

38. A terminal according to claim **36** in which the files are allocated addresses according to a predetermined algorithm and the terminal includes means operable for a said request message to calculate, in accordance with said algorithm, an address for inclusion in the request message.

* * * * *