

Method for playing on a player of a client device a content streamed in a network

FIELD OF THE INVENTION

The present invention relates to a method for playing a content streamed for example in a peer-to-peer network.

BACKGROUND OF THE INVENTION

“Streaming” designates a “direct” audio or video stream playing technique, that is while it is recovered from the Internet by a client device. Thus, it is in contrast with downloading, which requires to recover all the data of the audio or video content before being able to play it.

In the case of streaming, storing the content is temporary and partial, since data is continuously downloaded in a buffer of the client (typically the random access memory), analyzed on the fly by its processor and quickly transferred to an output interface (a screen and/or loudspeakers) and then replaced with new data.

Traditionally, the content is provided by streaming servers (referred to as a content delivery network, or CDN). The client which desires to access the content sends a request to recover first segments thereof (by segment, it is intended a data block of the content, corresponding generally to a few seconds of playing). When there is sufficient data in the buffer to enable the beginning of the content to be played, playing starts. In the background, the stream downloading continues in order to uninterruptedly supply the buffer with the remaining part of the content.

However, it is noticed that this approach has limits if a great number of clients desire to play the same content simultaneously: the server is found to be saturated, being incapable of providing the content at a sufficient rate for playing to be fluid, and jerks occur.

Recently, an alternative strategy based on “peer-to-peer” (P2P) has been suggested, in which each client acts as a server for other clients: they are called

peers. A peer which has started playing the content can forward to others segments it has already received, and so on, hence, easier broadcasting regardless of the number of clients being interested. This strategy is described in the international application WO 2012/154287.

However, most players implement what is known as Adaptive BitRate (ABR) and this proves to be problematic when combined with P2P.

The general idea of the ABR is to allow the automatic variation of the quality of the recovered segments according to the "capacities" of a peer. More precisely, each segment is available at several quality levels corresponding to several bitrates, i.e. data rates. It is indeed to be understood that a segment of better quality has better resolution, less compression, more frames per second, etc., and is consequently larger than the same segment in lower quality, therefore, it is necessary to support a higher data rate.

During ABR streaming, for each segment an algorithm automatically determines according to a given logic (referred to as "ABR logic") the best quality that can be chosen, generally in view of two criteria which are the observed bandwidth and/or the buffer filling rate.

In the first case, if the algorithm judges that the estimated bandwidth is sufficient to support higher quality, then it will instruct the client to switch to this (or conversely to lower the quality if the bandwidth is too low). In the second case, the principle is to divide the buffer memory into different intervals, each interval corresponding to an increasingly higher quality as the filling of the buffer memory increases (or more and less if it decreases).

In both cases, even if the ABR algorithms have no fundamental incompatibility to be used in a P2P streaming context, the problem is that the ABR algorithms were designed to work in a simple streaming scenario, i.e. with all segments retrieved on request from the content server.

However, in practice, P2P streaming advantageously performs "pre-fetching" (or "pre-buffering"), by downloading P2P segments into a dedicated P2P cache before the player actually requests them. Indeed, the objective of P2P streaming is to request as little as possible (and as a last resort) from the original content server: a direct request from a segment to this server is only made if there

is a risk that there are no more segments in the video buffer and that playback is interrupted ("re-buffering"), otherwise there is a maximum count on the P2P network.

We are thus left from the point of view of the player with extremely high apparent bandwidth since segments can be loaded into the buffer memory from the P2P cache a fraction of a second after they have been requested. In addition, the filling rate of the video buffer is artificially high.

This causes the ABR's uncontrolled decisions to increase the quality if the current quality is not the maximum quality, regardless of the actual network capacity, the quality that it may not necessarily be able to support.

To avoid unstable oscillations in the quality of the stream or even repeated interruptions of playing, and numerous and unnecessary requests to the content server, it has been astutely proposed in the applications FR1903195 and EP20305202 to introduce an artificial response delay before delivering a segment to the player to control the ABR algorithm, and how to choose the optimal response delay without having access to the ABR logic of the player.

This method is very satisfactory, but may still be improved.

Indeed, the normal pre-fetching process leads the P2P cache to keep a list of segments on the current bitrate, and, even if the ABR logic is under control, at each time the ABR switches to another bitrate these segments will be discarded, and another list of segments has to be fetched on the new bitrate. This scenario, if it happens frequently, consumes the P2P resources inefficiently, and costs requesting more data from CDN.

Moreover, in challenging bandwidth conditions, this may lead also to quality of experience (QoE) degradation due to the playback corruption that may happen in the case of switching to CDN when the P2P network itself is not good enough.

It would, therefore, be desirable to have a more efficient, universal and reliable pre-fetching strategy under any ABR algorithm in a P2P streaming context.

The present invention improves this situation.

SUMMARY OF THE INVENTION

For these purposes, the present invention provides according to a first aspect a method for playing on a player of a client device a content streamed in a network, said content consisting of a sequence of segments available in a plurality of quality levels, the player being configured so as to choose the quality level of the segments as a function of at least one parameter representative of a segment reception rate, according to an Adaptive BitRate, ABR, logic of the player; the client device comprising a first buffer for storing segments in a format adapted for transferring within the network, the method being characterized in that it comprises performing by a processing unit of the client device:

- (a) receiving from the player a request for a current segment at a first quality level;
- (b) determining that the player will request according to its ABR logic a next segment at a second quality level after said requested current segment is provided from the first buffer memory, using a model predicting the ABR logic of the player;
- (c) if said next segment is not present at said second quality level in the first buffer memory, fetching said next segment at said second quality level from the network.

Preferred but non limiting features of the present invention are as it follows:

Step (b) comprises predicting said second quality level with the model predicting the ABR logic of the player, as a function of said at least one parameter representative of a segment reception rate.

Said second quality level is predefined, step (b) comprising estimating an optimal response delay such that providing the requested current segment at the expiration of said optimal response delay will cause the player to request according to its ABR logic a next segment at said second quality level, as a function of said model predicting the ABR logic of the player.

Said model is trained from a database of training examples each associating a vector of measured parameters representative of a segment reception rate with the corresponding quality level subsequently chosen by the player according to its ABR logic.

Said ABR logic is defined by a first function of said at least one parameter representative of a segment reception rate, said model approximating the first function.

The client device further comprises a second buffer for storing segments in a format adapted for being played by the player, said current segment being provided at step (c) to said second buffer.

The requested current segment is provided from the first buffer memory at the expiration of said estimated optimal response delay.

Said parameter representative of a segment reception rate is a buffer level of the second buffer and/or a bandwidth.

Step (c) comprises, if said next segment is at least partially present, preferably entirely present, at said second quality level in the first buffer memory, performing again steps (b) and (c) for the second next segment.

Step (c) comprises, if said next segment is only partially present at said second quality level in the first buffer memory, fetching said next segment at said second quality level directly from a content server of the network.

The method comprises a further step (d) of verifying that a request for the next segment at the second quality level is received from the player.

According to a second aspect, the invention provides a device for playing on a player a content streamed in a network, said content consisting of a sequence of segments available in a plurality of quality levels, the player being configured so as to choose the quality level of the segments as a function of at least one parameter representative of a segment reception rate, according to an Adaptive BitRate, ABR, logic of the player; the client device comprising a first buffer for storing segments in a format adapted for transferring within the network, the client device being characterized in that it comprises a processing unit implementing:

(a) receiving from the player a request for a current segment at a first quality level;

(b) determining that the player will request according to its ABR logic a next segment at a second quality level after said requested current segment is

provided from the first buffer memory, using a model predicting the ABR logic of the player;

(c) if said next segment is not present at said second quality level in the first buffer memory, fetching said next segment at said second quality level from the network.

According to a third and a fourth aspect the invention provides a computer program product comprising code instructions to execute a method according to the first aspect for playing on a player of a client device a content streamed in a network; and a computer-readable medium, on which is stored a computer program product comprising code instructions for executing a method according to the first aspect for playing on a player of a client device a content streamed in a network.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other objects, features and advantages of this invention will be apparent in the following detailed description of an illustrative embodiment thereof, which is to be read in connection with the accompanying drawings wherein:

- Figure 1 represents an architecture for implementing the method according to the invention;
- Figure 2 illustrates a preferred embodiment of the method according to the invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Architecture

In reference to **Figure 1**, the invention relates to a method for playing a content streamed within a network 1 (advantageously within a peer-to-peer network 10 of client device 11, 12) using a trained model for predicting the ABR

logic of a player of a client device 11, advantageously trained according to a dedicated training method.

Network 1 is herein a large scale telecommunications network and in particular the Internet. This network 1 comprises the peer-to-peer network 10 of client devices 11, 12. Each client device 11, 12 is typically a personal computing device such as a smartphone, a PC, a tablet, etc. connected to the network 1, having a data processing unit 110 such as a processor, an interface for playing the content, and a storage unit such as a random access memory and/or a mass memory.

Playback is implemented by a player, that is to say an application executed by the data processing unit 110, which can be of a varied nature, for example a dedicated application, an internet browser in particular HTML5 compatible, an operating system module, etc. Note that the player may be defined by a name and a version.

We will assume in the following description that the player is "as is", i.e. not modified for the implementation of this process, or even for P2P streaming. In particular, the player implements an adaptive bitrate (ABR) logic, in other words said content to be played consists of a sequence of segments available in a plurality of quality levels and the player is able to decide autonomously which quality level to request, in accordance with this ABR logic. The various quality levels correspond to different bitrates, that is to say a variable volume of data per unit of time (and thus per segment). We naturally understand that better quality content requires a higher bit rate.

More details will follow regarding the concept of ABR logic, it is only to be understood that in the context of the present method it is not necessary that the ABR logic is controllable or even known: the present method is completely universal and can be adapted to any player implementing any ABR logic on the basis of any criteria. It will be assumed that the ABR logic is predefined and that the client software (see below) only undergoes it.

Furthermore, the client device 11 (and more precisely its storage unit) has two buffers M1 and M2, typically two zones of a random access memory, each being able to store (in a different way as will be seen) all or part of the content temporarily (by temporarily, it is meant that the segments are deleted from this

memory shortly after they have been played: they are not stored in the long term as is the case for a direct downloading). As will be seen later, in the preferred case of playing via a browser, all the segments are typically deleted (i.e. the buffers are reinitialized) at the latest when the browser or tab in which the video is played is closed.

The first buffer M1 is called "peer-to-peer cache". It stores segments under a so-called "raw" format. By raw segments, it is meant a format adapted for transferring within the network 1, in particular within the peer-to-peer network 10, but not adapted for playing on the device 11.

The second buffer M2 is called "video buffer". It stores segments under a so-called "converted" format. By converted segments, it is meant converted from the raw segments under a format adapted for playing on the device 11, but not adapted for transferring within the peer-to-peer network 10.

As explained in the introductory part, these devices 11, 12 are "peers" (also called "nodes") of the peer-to-peer network 10.

By "client devices 11, 12 of a peer-to-peer network 10", it is meant devices connected in the network 1 by a peer-to-peer network protocol. In other words, the data processing units for each peer implement a particular program (client software, referred to as "peer agent", PA), which can be integrated with the player (for example an extension of a web browser), be a dedicated application, or even be embedded into any other software (for example the operating system of an internet access box, or a multimedia box, i.e. a "Set-top box"), for using the peer-to-peer. The present method is mainly implemented via this client software. In the following description, it will be assumed that the client software is in communication with the player so as to provide it with segments, while operating independently. More precisely, we understand that the role of the player is the playing in itself, i.e. the rendering of the segments, while the role of the client software is simply obtaining the segments for the player, the client software undergoing the operation of the player, and in particular its ABR logic.

As explained, a peer-to-peer network, or P2P, is a decentralized sub-network within the network 1, wherein data can be directly transferred between two client devices 11, 12 of the peer-to-peer network 10, without passing through

a central server. Thus, it enables all the client devices 11, 12, to play both the role of client and server. The peers 11, 12 are thus defined as “seeders” (or data suppliers) and/or “leechers” (or data receivers).

Said content, which is in particular an audio or video content, that is a media of some length, consists of a sequence of segments (called a “playlist”) stored in data storage means of a server 2 connected to the peer-to-peer network 10. The segments have a predetermined length, typically one or two seconds of the content, but it can range from a fraction of a second to about ten seconds. All the segments of a given content have generally the same length.

The server 2 is a content server, advantageously present in network 1 and connected to the peer-to-peer network 10. In other words, this is one (or more) server(s) of the Internet network 1 providing the segments of various contents in accordance with a given streaming protocol. For example, the HLS (“HTTP Live Streaming”) will be mentioned, in which segments are “ts” files, listed in a “m3u8” playlist file. HLS involves the MPEG2 or the fragmented MP4 format for the content. DASH, Smooth streaming, or HDS streaming protocols will also be mentioned. The raw segments may be shared between peers via a protocol of the WebRTC type.

The server 2 is the primary source of the segments, insofar as initially no peer has the content (before a first transfer of the server 2 to this peer 11, 12). The contents are either at the very beginning stored integrally on the server 2 (case of the VOD previously discussed), or generated in real time (case of the live streaming), and in the latter case, the list of segments making it up dynamically changes over time.

Live streaming proposes to broadcast in real time contents associated with “live” events, for example concerts, meetings, sports events, video games, etc., which are simultaneously happening. With respect to streaming of already integrally existing content as a film, a live streaming broadcast content is actually generated gradually as the associated event happens. Technically, as in the case of a live event on TV, such content can only be broadcast with some delay, which the user wishes to be as small as possible. This delay is typically in the order of one minute, but can go down to about twenty seconds. Thereby, a playlist of only a few segments (at most some tens) is available at each instant, the segments of

this list being dynamically renewed in accordance with a turnover: as the event happens, new segments are created, "age", are received and played by clients (at the end of the expected delay), and finally exit the list.

In the latter case (live streaming), the content should rather be seen as a continuous stream. The sequence of segments is thereby dynamic, that is it is regularly updated. Each time a new segment is generated, it is added at the end of the sequence, and the first segment of the sequence (the oldest) is deleted. All the others are offset according to a turnover mechanism which can be related to a FIFO ("first in, first out") list. The first segment of the list (the oldest one) can be either a "live" or "past" segment. The "live" segment is the segment at the playing edge, and thus, the segments are deleted from the playlist as soon as they are played. The "past" segment exists when the content server 2 accepts that the content is played with some delay e.g. DVR (Digital Video Recorder) and other platforms that allow live streaming with up to a 2h delay.

The present method may be implemented in any context.

The peer-to-peer network 10 is also connected to a peer management server 3 called a "tracker". The tracker 3 has data processing means and storage means. It coordinates exchanges between peers 11, 12 (by controlling the client software implemented by each of the client devices 11, 12), but it is not directly involved in data transfer and does not have a copy of the file.

As explained, a dedicated method for training the model for predicting the ABR logic of the player may be implemented, either by the processing unit 110 of a client device 11 (or of another client device 12) or directly by the peer management server 3.

As it will be explained, the equipment performing the training has to store a training database made of data associated to a plurality of training segments already provided to the player (pairs of a vector of parameters representative of a segment reception rate measured when said training segment has been requested to the player and the corresponding quality level subsequently chosen by the player according to its ABR logic).

Note that there may be as many models (and databases) as types and versions of players, and the model for a given player may be learnt by a client

device 11, 12 or the server 3, and provided to all the client devices 11, 12 (directly from the server 3, or propagated as P2P messages) for predicting the ABR logic of this particular player at each client device 11, 12. Note that each client device 11, 12 may choose to receive only models corresponding to players it implements (and for example removes the previous model if there is a new version of a given player).

ABR logic

As already explained, the player of the client device 11 is configured so as to choose the quality level of the segments as a function of at least one parameter representative of a segment reception rate, according to an ABR logic of the player.

In any case, the ABR logic can be defined by means of a first function making it possible to calculate the quality level to be chosen (the bitrate) as a function of said at least one parameter representative of a reception rate of segments. More precisely, said first function is generally called by the player at each segment received, and the output is the quality level at which the next segment will be requested. Note that said output can be expressed in particular as an integer level number (for example between 1 and L, where 1 is the worst quality and L is the best quality or the opposite), or directly as a bitrate value (either a discrete value chosen among a possible bit rate values, or a continuous bitrate value). Said first function is supposed to be a "black box".

It is understood that said parameter representative of a segment reception rate is a monitored parameter, which can be any parameter illustrating the capacity of the device 11 and / or of the network 10 to receive the segments "fast enough". As mentioned, the known ABR logics generally use as a parameter a buffer level of the second buffer memory M2 (either in value, i.e. in seconds or in number of segments, or in rate) and/or a bandwidth (i.e. the data reception rate observed).

In other words, the player monitors the bandwidth and/or the buffer level, and consequently makes decisions as to whether or not to modify the quality level of the segments required.

Note that other parameters are sometimes taken into account, such as device capabilities (including the CPU/GPU load and decoding capabilities, available memory, screen size, etc.) and/or user geographical location.

Thus, there are three main classes of ABR logic:

- The “BB” class, for buffer-based ABR logic;
- The “RB” class, for rate-based ABR logic;
- The “H” class, for hybrid (buffer-rate-based) ABR logic.

Note there might be further classes. The following specification will take the example of these three classes but the skilled person will understand that the present method is not limited to any set of possible ABR logic classes.

Model predicting the ABR logic

The present method uses a model predicting the ABR logic, i.e. approximating the above-mentioned first function defining the ABR logic, regardless of the actual class of the ABR logic.

Said model may be predefined, but preferably machine learning (ML) algorithms can be used to train the model, as proposed in the application EP20305202.

Mathematically, a vector of parameters can be built for any given segment (once requested and received by the player), where the vector of parameters is representative of a segment reception rate measured when said given segment has been requested by the player (i.e. an “input vector” x^m for a segment $m \in \llbracket 1; M \rrbracket$) associated to the corresponding quality level subsequently chosen by the player according to its ABR logic (i.e. the “scalar output” $y^m = [br_m]$, br_m being the bitrate decision for the segment m).

The idea is to include in the input vector x^m any possible parameter representative of a segment reception rate so as to encompass any possible ABR class and logic, such as b_m the buffer level at which the segment m is requested, s_m the segment size, t_m the segment download time, bw_m the measured bandwidth of segment m , bw_{m-1} the measured bandwidth for the previous segment, etc.

The pair of an input vector x^m and the corresponding scalar output y^m is denoted a training example, and a database of training examples may be built for performing a machine learning algorithm so as to train a model. It is to be understood that, as explained, each training example corresponds to the actual reception of a given segment (that can be referred to as a training segment) by the player. In other words, each training example associates a vector of parameters representative of a segment reception rate measured when a training segment has been received by the player, and the corresponding quality level subsequently chosen by the player according to its ABR logic for requesting the next segment.

Said model can be defined as the relation between the input and the output, in particular a “hypothesis” h_θ parameterized with a vector θ of model parameters such that for each input vector x^m the value $h_\theta(x^m)$ is as close as possible to y^m .

Thus, by measuring in real time the current parameters representative of a segment reception rate and generating a current input vector x^i , the hypothesis $h_\theta(x^i)$ can be used to predict the output $\hat{y}^i$ which is the bitrate at which next segment will be requested by the player.

Thus, the present method advantageously comprises an initial step (a0) of training a model from said database of training examples each associating a vector of measured parameters representative of a segment reception rate (i.e. measured for a given segment, when it had been requested by the player) with the corresponding quality level subsequently chosen by the player according to its ABR logic (when requesting the next segment).

Note that any type of model and any kind of machine learning algorithm may be used.

Preferably, the model is a linear function approximating said first function (linear regression), and it is learnt by linear least square (LLS) techniques, in particular ordinary least squares (OLS) techniques, but the skilled person could use other models (notably polynomial, non-linear, etc.) and other machine learning techniques (Bayesian, k-Nearest Neighbors, Support Vector Machine, etc.).

As explained, step (a0) may be performed locally by the client 11, or in a centralized way at the server 3. In any case, the training examples may be transmitted within the network 1 for constituting the training database. For example, raw data may be collected from various clients 11, 12 at the server 3, wherein processed training data (such as the training matrix X and the training vector Y) is built, and possibly sent back.

Further, as explained in the already cited application EP20305202, step (a0) preferentially comprises training a plurality of models (in parallel), one for each class of ABR logic, so as to safely remove the redundant features and keeping only the actual inputs of the ABR logic as implemented by the player (ABR logics of BB, RB and H classes use different input variables). In our example, there are $K=3$ classes (BB, RB and H) so that 3 models are trained.

For a given player, only one of the K models is actually true (i.e. properly predicts the ABR logic of the player). Therefore, step (a0) advantageously further comprises verifying the K models so as to select the appropriate one (the others are discarded) in particular by building a test set (i.e. keeping some pairs (x^m, y^m)) to check the categorical accuracy on said test set.

Finally, the selected model might be shared with any device 11, 12 so as to be used at large scale. Propagation of this model amongst peers may be done either from the server 3 or directly by P2P. Note that any peer receiving a model may test it and/or refine it by restarting a new training step (a0).

Controlling the ABR

In the following description, we focus on client device 11 which is trying to retrieve the content from other devices 12 and / or the server 2, that is to say, the first buffer memory M1 already stores at least one raw segment, in at least one quality level, if possible a sub-sequence of the sequence constituting the content.

It is supposed that the model suitable for the player (i.e. predicting the ABR logic of said player) is already trained, selected and available to the device 11.

The method then begins with the implementation by the processing means 110 of the device 11 of a step (a) of receiving a request for a segment (referred to as "current segment"), in practice the next segment to be put in the second

buffer memory M2 (not necessarily the next segment to be played, there are normally buffered advance segments). Said request is received from the player, and defines the quality level which is required for the requested segment, i.e. the bitrate (by applying ABR logic), referred to as “first quality level”.

It is assumed that said segment is at least partially available at this stage (i.e. at least a fragment) in the first buffer M1, in the first quality required by the player. If this segment / segment fragment was in another quality, it would have to be retrieved again, generally directly from the content server 2 because we are running out of time.

Step (a) includes, if necessary, the “measurement” of said at least one parameter representative of a segment reception rate.

In a following step (b), the trained model is used to determine that the player will request, according to its ABR logic, a next segment (with respect to the current segment, i.e. segment $m + 1$ if current segment is segment m) at a second quality level after said requested current segment is provided from the first buffer memory M1.

Here “determine” has to be understood in a broad way, and simply means being able to know the next ABR decision. This step may be only “passive”, i.e. the client 11 does not try to control the ABR logic and simply undergoes it, or “active”, meaning that there is an “intended” second quality level. In this latter case, determining that the player will request according to its ABR logic a next segment at a second quality level means ensuring that this second quality level is *in fine* reached, or at least attempting to. The present invention will not be limited to any case, it is sufficient that at the end of step (c) the second quality level, i.e. the quality level for the next segment, is at least estimated.

In the “passive” embodiment, the second quality level may be unknown, and thus step (b) simply comprises predicting said second quality level using the model. Note that the second quality level may be the same as the first quality level.

In the more advantageous “active” embodiment, the second quality level may be as explained a goal to reach by actively controlling the ABR logic of the player using a response delay, as proposed in the application FR1903195.

Therefore, in this preferred embodiment, said second quality level is predefined and step (b) comprises estimating, for this second quality level, an optimal response delay such that providing the requested current segment at the expiration of said optimal response delay will cause the player to request according to its ABR logic a next segment at said second quality level.

In other words, we intend to control the ABR logic so as to “force” it to request the next segment at the second quality level. By optimal response delay it is meant a response delay suitable for causing the ABR logic to request the second quality level (thus the optimal response delay is not necessarily unique, and generally there is a “range” of optimal response delay). Mathematically, for the next segment $m + 1$, we have to trigger an input vector x_{m+1} of the model such that $\hat{y}_{m+1} = h_{\theta}(x_{m+1})$ is the second quality level expected to be requested.

To this end, it is first important to understand the relation between the response delay and the input variables of the model: if p is the segment duration (generally fixed) and d_m is the response delay to apply for the current segment m , we have:

- $b_{m+1} = b_m + p - d_m$, because the buffer will be gradually emptied by the playing (useful for BB-models and H-models);
- $bw_{m+1} = \frac{s_{m+1}}{t_{m+1}} \approx \frac{s_m}{d_m}$, because the delay may be translated as download time (the actual transfer time is nearly zero as the segment is already downloaded in the first buffer M1) and s_m is often nearly constant (useful for RB-models and H-models).

Other parameters of the input vector x_{m+1} may be estimated from measurements and parameters of the current vector x_m . Note that some parameters estimated for calculating x_m might have been estimated from x_{m-1} , and their value corrected (b_m can be measured).

The application EP20305202 explains how to efficiently “reverse” the model to determine the input vector x_{m+1} such that $y_{m+1} = h_{\theta}(x_{m+1})$. It is to be understood that solving such an optimization problem is within the grasp of a skilled person.

In the case where it is only a fragment of the requested segment which has been retrieved from the P2P network (it is said that the segment is available

in an incomplete manner), preferably the estimated optimal response delay is modified according to the length of the fragment so as to reflect the fact that only a fragment of the optimal response delay should actually be applied. Indeed, the second buffer M2 can only be provided with complete segments and not fragments, and the idea is to provide the segment in full after a shorter response delay reflecting the fact that there will already be an implicit waiting delay corresponding to the time to complete (finish retrieving) this segment in the first buffer M1. Thus, step (b) may include modifying the estimated optimal response delay as a function of an estimated duration necessary to finish retrieving the segment.

For example, we could apply the formula $d'_m = d_m - tdw_m$, where d'_m is the modified optimal response delay and tdw_m is the estimated time needed to finish retrieving the segment. So, waiting for time tdw_m plus applying d'_m before delivering the full segment is equivalent to applying d_m , so the overall delay remains the same.

In the “passive” embodiment wherein there is not any response delay applied, the remaining time tdw_m can be directly used as download time for calculating bw_{m+1} .

Note that the “active” embodiment is not limited to the case of the “optimal” response delay. In other words, a response delay which is not the optimal one could be actually applied (for example because the remaining time tdw_m was longer than the optimal response delay, or for any other reason to use a different response delay – the response delay could even be predetermined), and in such case step (b) could still comprise predicting the “real” second quality level using the model, after having determined the actual response delay that will be applied and taken into account the impact of said actual response delay on the input variables of the model.

Pre-fetching

In a very unusual way, the present method comprises a step (c) of, if said next segment is not present at said second quality level in the first buffer memory

M1), fetching it from the network 1 (i.e. performing "pre-fetching", as we are at least one segment ahead).

In other words, while the next segment should have been normally fetched at the first quality level (which is still the current quality), here the switch to the second quality level is anticipated, so as to save bandwidth. Indeed, if the segment was fetched at the first quality level, the first memory M1 would have to discard it if the second quality level was different from the first quality level. As the second quality level is taken into account in advance, bandwidth is saved.

By "fetching", it is meant obtaining said segment from the network 1, by download to the P2P cache M1 from another peer 12 and/or from the content server 2, as it will be explained. Note that said next segment may actually have already been entirely downloaded (case "F-HIT", see below), in particular if the second quality level is the same as the first quality level, so that no further action is necessary, but in any case step (c) at least comprises checking whether said next segment is at least available at said second quality level in the first buffer memory M1.

Advantageously, step (c) comprises the three following cases:

- MISS: the next segment is not present (at all) in the first memory M1, i.e. it has to be fetched;
- F-HIT (full hit): the next segment is entirely present in the first memory M1, i.e. it has already been fetched;
- P-HIT (partial hit): the next segment is only partially present in the first memory M1, i.e. fetching is ongoing.

In the case MISS, step (c) comprises fetching said next segment at said second quality from the network 1. Note that it may be necessary to request it from the content server 2, but we may still have the time to use P2P.

In the case F-HIT, everything is perfect (no fetching is needed), so the second next segment $m + 2$ may be considered: steps (b) and (c) can be repeated for said second next segment (i.e. determining that the player will request according to its ABR logic a second next segment at a third quality level after said next segment is provided from the first buffer memory M1, using the model predicting the ABR logic of the player; and, if said second next segment is not present at said third quality level in the first buffer memory M1, fetching said

second next segment at said third quality level from the network 1), and so on for a third next segment, etc.

In the case P-HIT, there are multiple options:

- A first possibility is to do nothing in particular, i.e. to consider that the segment is being downloaded in the suitable quality level (no bandwidth will be wasted) and to simply wait for the download to be completed. Thus, in the next occurrence of step (b) with the next segment (as new "current" segment), a further response delay will consider the reminding download time tdw_{m+1} ;
- A second possibility is to directly complete the rest of the segment from the content server 2. This option is good for two reasons:
 - o it updates the measurements relative to the content server 2 (CDN measurements) from time to time,
 - o it improves the response delay work by better controlling the next segments, i.e. it is guaranteed that the segment will be completed (i.e. fully available in the first memory M1) by the time it is requested by the player in the next occurrence of step (a), so that any necessary response delay d_{m+1} could be applied so thus we keep the smooth work of ABR controlling and pre-fetching of the following segments.

Simultaneously, step (c) comprises providing the current segment (segment m) in response to the request, from the first buffer M1.

If any response delay is to be applied (in particular if an optimal response delay has been estimated at step (b)), said segment is advantageously provided at the expiration of said response delay. By "provided at the expiration of said response time" is meant so that the player does not have it before the expiration of the response delay (at best at the time of expiration, or even only after in some cases, see below). Most often, the segment is transmitted suddenly when the response delay expires, but it will be understood that it is quite possible to "stream" it within device 11, i.e. to transmit it from the first buffer M1 gradually (piece after piece) so that the last piece is transmitted (at the earliest) when the optimal response delay expires (the optimal response delay is then a

"transmission time of the last bit of the segment"). Indeed, although only complete segments are playable, some players can accept sub-segments of the segment. Note that such a progressive transmission does not change anything since as long as the segment is not fully received it is not available by the player and therefore not considered to be provided, but makes it possible to facilitate bandwidth measurements.

In the case where only a fragment of the segment was available in the first buffer M1 and the response time has been modified according to an estimated duration necessary to finish recovering the segment, normally the segment is also supplied to the step (b) at the end of the modified response time. As explained, although the supply can be fragmented, one should not confuse sub-segments of a complete segment (which correspond to consecutive pieces of segment obtained from a completely downloaded segment) and an incomplete segment (in which only certain parts of the data, most often corresponding to disparate pieces, have been downloaded). Only a segment completely available in the first buffer M1 can be provided (progressively if necessary) in response to the request (and not a fragment), so that if the download takes longer than expected, the segment may not be fully available until after the modified response delay has expired. Thus, as explained the complete segment is provided at the earliest at the expiration of the modified optimal response delay (i.e. not before), but possibly after. In practice, the complete segment is provided when the following two conditions are satisfied: the segment is completely available (its download is complete), and the modified optimal response delay has expired.

In all cases, the segment is preferably provided to the second buffer M2, and as such step (c) can comprise the conversion into a format suitable for playing said segment. This consists in transforming the raw segment into a converted segment, which can be read by the player of the device 11, unlike the raw one.

For example, if the player is the built-in player of an HTML5 compatible browser, the conversion consists of injecting the segment's video data using the Media Source Extension API of the browser

Naturally, step (c) advantageously comprises simultaneously playing a previous segment (for instance segment $m - 1$, but possibly an even older

segment) stored in the second buffer memory M2, so that the segments need to be renewed. The segment retrieved in step (c) will soon be read in turn.

We can now repeat steps (a) to (c) as long as the playing lasts: the next segment is now the new current segment, and the second quality level is now the new first quality level (possibly because of the application of the optimal response delay which has forced said second quality level as predicted)

In other words, a new occurrence of step (a) consists in receiving from the player a request for the next segment at the second quality level. Again, it could be determined that the player will request according to its ABR logic a next segment (i.e. segment $m + 2$) at a third quality level, and in particular a new optimal response delay (such that providing the requested next segment at the expiration of said new optimal response delay will cause the player to request according to its ABR logic a next segment at said third quality level) could be estimated then applied, etc.

Note that the method may include a step (d), at the end of step (c), for verifying the prediction. In other words, it is verified that a request for the next segment at the second quality level is received from the player. This step (d) is typically included in the next occurrence of step (a), wherein the request for the next segment is actually received. Verification simply involves comparing the predicted/intended second quality level with the quality level actually requested for the next segment (by the ABR logic).

Device and computer program product

According to a second aspect, the invention concerns the device 11 for performing the previous described method for playing a content (streamed in a peer-to-peer network 10 of client devices 11, 12) on a player of the device 11 configured so as to choose the quality level of the segments as a function of at least one parameter representative of a segment reception rate, according to an ABR logic of the player.

This device 11 comprises as explained:

- a first buffer M1 (P2P cache) for storing segments in a format adapted for transferring within the network 1 and in particular the peer-to-peer network 10;
- Preferably a second buffer M2 (video buffer) for storing segments in a format adapted for being played by the player;
- a processing unit 110.

The processing unit 110, typically a processor, is implementing the following steps:

(a) receiving from the player a request for a current segment at a first quality level;

(b) determining that the player will request according to its ABR logic a next segment at a second quality level after said requested current segment is provided from the first buffer memory M1, using a model predicting the ABR logic of the player;

(c) if said next segment is not present at said second quality level in the first buffer memory M1, fetching said next segment at said second quality level from the network 1.

In a third and fourth aspect, the invention concerns a computer program product comprising code instructions to execute a method (particularly on the data processing unit 110 of the device 11) according to the first aspect of the invention for playing on a player of a client device 11 a content streamed in a peer-to-peer network 10 of client devices 11, 12, and storage means readable by computer equipment (memory of the device 11) provided with this computer program product.