



ELSEVIER

Contents lists available at SciVerse ScienceDirect

Signal Processing: *Image Communication*journal homepage: www.elsevier.com/locate/image

Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network

Chenghao Liu^{a,*}, Imed Bouazizi^b, Miska M. Hannuksela^{b,♣}, Moncef Gabbouj^{a,♣}

^a Department of Signal Processing, Tampere University of Technology, Finland

^b Nokia Research Center, Tampere, Finland

ARTICLE INFO

Keywords:

Rate adaptation
Dynamic Adaptive Streaming over HTTP
DASH
Adaptive HTTP Streaming
CDN

ABSTRACT

Recently the 3rd Generation Partnership Project (3GPP) and the Moving Picture Experts Group (MPEG) specified Dynamic Adaptive Streaming over HTTP (DASH) to cope with the shortages in progressive HTTP based downloading and Real-time Transport Protocol (RTP) over the User Datagram Protocol (UDP), shortly RTP/UDP, based streaming. This paper investigates rate adaptation for the serial segment fetching method and the parallel segment fetching method in Content Distribution Network (CDN). The serial segment fetching method requests and receives segments sequentially whereas the parallel segment fetching method requests media segments in parallel. First, a novel rate adaptation metric is presented in this paper, which is the ratio of the expected segment fetch time (ESFT) and the measured segment fetch time to detect network congestion and spare network capacity quickly. ESFT represents the optimum segment fetch time determined by the media segment duration multiplied by the number of parallel HTTP threads to deliver media segments and the remaining duration to fetch the next segment to keep a certain amount of media time in the client buffer. Second, two novel rate adaptation algorithms are proposed for the serial and the parallel segment fetching methods, respectively, based on the proposed rate adaptation metric. The proposed rate adaptation algorithms use a step-wise switch-up and a multi-step switch-down strategy upon detecting the spare networks capacity and congestion with the proposed rate adaptation metric. To provide a good convergence in the representation level for DASH in CDN, a sliding window is used to measure the latest multiple rate adaptation metrics to determine switch-up. To decide switch-down, a rate adaptation metric is used. Each rate adaptation metric represents a reception of a segment/portion of a segment, which can be fetched from the different edge servers in CDN, hence it can be used to estimate the corresponding edge server bandwidth. To avoid buffer overflow due to a slight mismatch in the optimum representation level and bandwidth, an idling method is used to idle a given duration before sending the next segment. In order to solve the fairness between different clients who compete for bandwidth, the prioritized optimum segment fetch time is assigned to the newly joined clients. The proposed rate adaptation method does not require any transport layer information, which is not available at the application layer without cross layer communication. Simulation results show that the proposed rate adaptation algorithms for the serial and the parallel segment fetching methods quickly adapt the media bitrate to match the end-to-end network capacity, provide an advanced convergence and fairness between different clients and also effectively control buffer underflow and overflow for DASH in CDN. The reported simulation results demonstrate that the parallel rate adaptation outperforms

* Corresponding author. Tel.: +358 509349231.

E-mail address: chenghao.liu@tut.fi (C. Liu).

♣ EURASIP member.

the serial DASH rate adaptation algorithm with respect to achievable media bitrates while the serial rate adaptation is superior to the parallel DASH with respect to the convergence and buffer underflow frequency.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

Recently, Hypertext Transfer Protocol (HTTP) has been widely used for the delivery of real-time multimedia content over the Internet, such as in video streaming applications. Unlike the use of Real-time Transport Protocol (RTP) over User Datagram Protocol (UDP), HTTP [1] is easy to configure and is typically granted traversal of firewalls and network address translators (NATs), which makes it attractive for multimedia streaming applications. Krasic et al. [2], Wang et al. [3] and Kim and Ammar [4] reported that short-term transmission rate variation in HTTP/TCP can be smoothed out by receiver side buffering. Thus, commercial solutions, such as Microsoft Smooth Streaming [5] and Adobe Dynamic Streaming [6], have been launched as well as standardization projects have been carried out. Adaptive HTTP streaming (AHS) was first standardized in Release 9 of 3GPP packet-switched streaming (PSS) service [7]. MPEG took 3GPP AHS Release 9 as a starting point for its newly published MPEG DASH standard [8]. 3GPP continued to work on adaptive HTTP streaming in communication with MPEG and recently published the 3GP-DASH (Dynamic Adaptive Streaming over HTTP) [9]. MPEG DASH and 3GP-DASH are technically similar and are therefore collectively referred to as DASH in this paper. In DASH, the client continuously requests and receives small segments of multimedia content, denoted as media segments. To adapt the media bitrate to the varying network bandwidth, DASH allows clients to request media segments from different representations, each of which represents a specific media bitrate.

A DASH client might suffer from frequent interruptions and sub-optimum media bitrates without an efficient rate adaptation algorithm. A media bitrate higher than the sharable bandwidth would cause network congestion. As DASH typically uses Transmission Control Protocol (TCP) as transport layer protocol, network congestion causes TCP congestion control mechanism to become active, which may further result in a dramatic decrease in throughput. Hence, buffered media data can be drained-up much faster in DASH compared to the traditional RTP/UDP-based streaming in case of network congestion. On the other hand, i.e., when the media bitrate is lower than the sharable bandwidth, the media quality does not reach the optimum allowed by the available bandwidth. Many of the Internet's real-time video services use a progressive download approach and hence suffer from the above mentioned problems, such as frequent playback interruption and sub-optimum streaming quality. Many video services offer a set of pre-defined quality levels of a video clip to users for manual a-priori selection. If the bitrate of the selected representation turns out to be higher than the available end-to-end bandwidth, then the user will most probably experience playback interruptions and re-buffering events due to buffer underflow. Otherwise, if the bitrate of the representation is lower than the available network

bandwidth, then the user will consume the content at a sub-optimal quality. Since the client will download content faster than the playout rate, it could result in inefficient use of available bandwidth if the user stops watching the content. An efficient rate adaptation algorithm is required for DASH to solve the above problems, specifically, the frequent interruption in playback and sub-optimum streaming quality. However, these problems become even more challenging because of the difficulties in differentiating between the short-term throughput variation, incurred by the congestion control, and the average throughput changes due to more persistent bandwidth changes. Furthermore, the rate adaptation algorithm of DASH should take into account the infrastructure of the networks.

The research issues of the rate adaptation for DASH include the following aspects. First, the rate adaptation method must deploy a metric to determine if the bitrate of a specific representation matches the available end-to-end bandwidth. This metric is expected to distinguish between persistent throughput variation due to network bandwidth changes and the short-term throughput dynamics attributable to the TCP congestion control. In addition, the metric should identify network congestion and spare network bandwidth fast enough in order to react promptly and reach the optimum representation level as soon as possible, wherein the higher representation level represents the higher level of media bitrates and the lower representation level represents the lower level of media bitrates. Second, the rate adaptation algorithm should prevent frequent hopping between media representations, because frequent changes in the perceived media quality are likely to be annoying. In segment delivery over distributed networks, such as content distribution network (CDN), preventing too frequent representation switching becomes more challenging as segments can be transmitted through different routes having different bandwidths. Thirdly, the rate adaptation algorithm should avoid client buffer underflow and overflow. Buffer underflow causes playback interruptions and overflow can result in bandwidth waste. Fourthly, the rate adaptation algorithm for DASH should provide good fairness between different DASH clients, which compete for available bandwidth in the network. Finally, the media segment duration should be set appropriately in order to smooth the short-term HTTP/TCP throughput variation and to provide low enough adaptation latencies.

Two strategies for issuing HTTP requests in DASH have been proposed, namely the serial segment fetching method [10] and the parallel segment fetching method [11]. In the former, segments are requested and received one after another, whereas in the latter, media segments are requested in parallel using several TCP connections. This paper presents novel rate adaptation algorithms for both the serial and the parallel segment fetching methods.

The main contributions of this paper include the following two aspects. First, we propose a novel rate adaptation metric for DASH. The proposed rate adaptation metric compares the expected segment fetch time (*ESFT*) with the measured segment fetch time (*SFT*) to determine if the media bitrate matches with the current bandwidth. The *ESFT* is the optimum segment fetch time, which is estimated each time before requesting a segment based on (a) the segment duration multiplied by the number of parallel segment fetching threads (b) the remaining time period to fetch the next segment. The former is determined by the scheduling method of the serial and the parallel segment fetching methods, whereas the latter is decided by the real-time buffered media time. The rate adaptation metrics not only take into account the ideal segment fetch time of the serial and the parallel segment fetching methods but also consider the buffered media time. Hence, the proposed rate adaptation metrics can provide convergence in the representation level and prevent buffer underflow.

Second, we propose two novel rate adaptation algorithms for the serial and the parallel segment fetching methods in CDN, respectively. The proposed rate adaptation algorithms for the serial segment fetching method uses a single rate adaptation metric to determine a switch up/down of representation levels. The proposed rate adaptation algorithm for parallel segment fetching method deploys a sliding window to select the minimum rate adaptation metrics among multiple metrics. The minimum rate adaptation metrics is used to determine the switch-up/switch-down in order to match the media bitrates to the minimum bandwidth in the distributed edge servers of CDN. Thus, the sliding window strategy of the proposed rate adaptation algorithm for the parallel segment fetching method can efficiently prevent frequent hopping between representation levels. Furthermore, the proposed rate adaptation algorithms give a priority for newly joined DASH clients in order to share the available bandwidth fairly among DASH clients. In case of congestion, DASH clients consuming a higher representation level will quickly switch down to a lower level firstly in order to leave room for DASH clients having a lower level using the proposed rate adaptation algorithm. Furthermore, in order to efficiently use network bandwidth and memory resources for the users, a method of determining the idle time before sending an HTTP GET request for the next segment is presented, thus limiting the maximum amount of pre-fetching media.

The rest of this paper is organized as follows. Section 2 presents the related work to describe the literature works relating to the rate adaptation for DASH. Section 3 gives a brief overview of the terminology of DASH, and DASH infrastructure using a CDN. The serial and the parallel segment fetching methods are presented in Section 4. In Section 5, a novel rate adaptation metric for DASH is presented. Sections 6 and 7 provide further details on the two novel rate adaptation algorithms for the serial and the parallel segment fetching methods in CDN, respectively. The simulation results of the proposed rate adaptation algorithms for the serial and the parallel segment fetching methods in CDN are provided in Section 8. Finally, Section 9 concludes the paper.

2. Related work

In the last decade, several research works have been conducted in the rate adaptation for media streaming over TCP, called TCP-based streaming in this paper. In the scenario of the TCP-based media streaming, most of the research works focused on sender-driven rate adaptation. For example, Lam et al. [12] proposed a method to estimate client-side buffer occupancy in the server and to adapt media bitrates to maintain the client-side buffer occupancy above a certain threshold.

Rate adaptation for DASH is a relatively new research topic and thus only few research works have been carried out in this field. In DASH, decisions to adapt media bitrates are made at the client side where the client buffer occupancy is known and can be used directly for the rate adaptation. For example, in one of the earliest papers on adaptive streaming over HTTP by Färber et al. [13], rate adaptation decisions were made on the basis of media duration in the client buffer. However, client buffer occupancy based rate adaptation for DASH has the following drawbacks. First, this method can cause hopping in the representation level. As the choices of representation levels are limited, it is difficult to match the optimum media bitrate to the network capacity. Typically, the optimum media bitrates should be the highest bitrate, which is lower than the bandwidth. In such a scenario, the client buffer occupancy, which can be measured as buffered media time, will dramatically increase even if the rate adaptation reaches the optimum media bitrate. Hence, the rate adaptation cannot converge to the optimum representation level as the client buffer occupancy will change even when the rate adaptation reaches the optimum media bitrates. Second, the client buffer based rate adaptation can result in unfairness between different clients, which compete for bandwidth. For example, some clients can switch to a high-bitrate representation if adequate bandwidth is available, while other clients might not be able to fetch equally high-bitrate representation if the networks are already on the stage of saturation when they start streaming. Efficient rate adaptation algorithm should provide a fair allocation of the network bandwidth to clients. Kuschig et al. [14] presented a method of priority and deadline driven rate adaptation method for scalable video streaming over TCP using bandwidth estimation and client buffer estimation at the server. To stabilize TCP throughput over a given period of time and decrease quality fluctuation, relatively large groups of pictures (GOPs) are used to estimate TCP throughput by deploying the application layer based and TCP stack based bandwidth estimation techniques. As pointed out in [14], the accuracy of the bandwidth estimation method suffers from the large GOP, which is coped with the priority and deadline driven rate adaptation method proposed in the same paper. Recently, Akhshabi et al. [15] reported experimental evaluation of the rate adaptation in DASH. In [15], the experiment is performed using the Microsoft smooth streaming, Netflix player, and Adobe OSMF player to evaluate the internal rate adaptation algorithms with certain experimental methodology.

In our previous work [10], a rate adaptation algorithm for the serial segment fetching method was presented. In

[10], the ratio of the segment duration and the latest single segment fetch time was used to detect network congestion and spare network capacity. However, the rate adaptation algorithm in [10] has the following shortcomings. First, the rate adaptation method was developed for the case where segments are requested and received sequentially, and hence *ESFT* is determined in [10] to be equal to the media segment duration (*MSD*). However, for the parallel segment fetching method where the number of simultaneous HTTP transactions can be larger than one, *ESFT* should also be proportional to the number of parallel HTTP threads. Second, the buffered media time status of the client buffer is not taken into account in the rate adaptation metric proposed in [10], which causes buffer underflow due to the lower buffered media time. To prevent buffer underflow, the buffered media time should be used as the metric for determining the next segment media bitrates. Third, the rate adaptation method compares the ratio of the segment duration by the latest received single segment fetch time with the switch-up and switch down thresholds to adapt the representation level for the serial segment fetching method. In the parallel segment fetching method, multiple segments are received simultaneously, hence multiple segment fetch times should be used to decide whether or not to perform switching-up in order to improve convergence to a single representation level when the networks resource capacity is not changed severely. Fourth, the fairness issue between different DASH clients has not been discussed in [10].

DASH may employ the current web distribution network infrastructure, e.g., proxy caches and CDN. Currently, CDNs are widely used to deliver media and web data to end users through globally distributed edge servers. CDNs, such as Akamai, deploy Domain Name System (DNS) based request redirection technique to redirect a client request to an appropriate edge server among multiple edge servers with the predefined policies. Each client request is used to fetch a media segment in DASH. The detailed request redirection mechanism and the redirection frequency are described in Section 3.

Scheduling of HTTP GET requests for segment fetching plays a critical role in achievable streaming bitrates for DASH in CDN. However, the specific scheduling method of requesting segments has not been specified in DASH. Only the informative instructions of the client behavior are provided in both 3GP-DASH and MPEG-DASH. One common method is the serial segment fetching method, which requests and receives segments one after another. For the DASH in CDN, the serial segment fetching method might not yield the optimum streaming service to DASH clients since it only operates one HTTP thread at a time to deliver media segments over the distributed edge servers in CDN, which results in inefficient usage of the distributed network resources. The parallel segment fetching method can occupy network resources of CDN more exhaustively as multiple HTTP threads are used to deliver the media segments in parallel over distributed networks.

In our previous work [11], a parallel adaptive HTTP streaming method was presented. The method schedules HTTP GET segment requests in parallel at the DASH client, for efficiently utilizing the distributed networks resources

and improving the achievable streaming quality. Specifically, our previous paper presented a method, which (a) enables a receiver to request multiple segments in parallel, (b) provides a solution to maintain a limited number of HTTP sessions for receiving segments in parallel and to determine when to start a new HTTP session to request the next segment and (c) can adapt the received media bitrate while receiving previously requested segments. For the rate adaptation algorithm, [11] deployed a simple rate adaptation algorithm, which uses the buffered media time to determine the media bitrates, hence results in similar problems as described above for [13]. Since the focus was on improving achievable representation level, the simulation results in [11] were limited in reporting the average representation level for a single DASH client user. In this paper, we investigate the rate adaptation for the serial and parallel segment fetching in CDN.

In a multiple TCP connection based media streaming, requesting a set of media chunks using multiple TCP streams was presented in [16,17]. Multiple TCP connection based media streaming was previously presented in [18]. The work in [16,17] extended multiple TCP connections based media streaming to the request-response based internet video streaming. In the same papers, the authors presented analytical models for the total achievable TCP throughputs for the request-response based TCP streaming using multiple TCP connections. Although this paper uses the parallel HTTP requests, the aim and the underlying principles are totally different from those in [16,17], where the aim of multiple TCP connections was to provide resilience against short-term insufficient bandwidth using multiple TCP connections for a streaming application. In contrast, the target in our previous paper [11] was to fully utilize the distributed network resources, for example CDN, to improve HTTP-Streaming quality at the client. The underlying principle of the works [16–18] was that by allocating a certain bottleneck bandwidth to multiple TCP streams, the TCP throughput variation can be smoothed out and the achievable TCP throughput can be increased to some degree as multiple TCP streams do not observe packet losses at the same time. In contrast, [11] is inspired by the distributed infrastructure of CDN.

Evensen et al. [19] presented a method of deploying multiple heterogeneous interfaces to improve streaming quality. They proposed to divide a segment into multiple sub-segments dynamically and request sub-segments using distributed requests over multiple heterogeneous interfaces simultaneously. As [19] uses sub-segments, several techniques for partitioning segments into sub-segments were studied and the paper proposed a dynamic and static sub-segment partitioning method. In contrast, our previous paper [11] used segment as a fetching unit, which makes the parallel segments fetching method simpler compared to the parallel sub-segment fetching method [19]. The underlying principle of paper [11] is the efficient utilization of the distributed edge servers in CDN. On the contrary, [19] considered a device using multiple heterogeneous network interfaces, such as a smart phone equipped with the ability to transfer data through WLAN and HSDPA networks simultaneously.

3. Terminology and architecture of DASH in CDN

In this section, we shall first explain the terms and definition of DASH as specified in 3GPP PSS. Then, the end-to-end systems of DASH, including the DASH content provider, DASH server and DASH client, are described. In addition, the media distribution architecture in CDN is presented.

In DASH, the media presentation description (MPD) provides the necessary information for clients to establish a dynamic adaptive streaming over HTTP. The MPD contains the metadata, such as an HTTP-URL of each segment, to make GET segment request. The segment contains a certain duration of media data, and metadata to decode and present the included media content. Each representation consists of multiple media segments and represents the coding choice such as encoded bitrate, spatial and temporal resolution, etc. The representation is identified through a unique identifier (ID). The client can obtain the available number of representations and the corresponding properties by accessing the MPD.

Fig. 1 shows a media distribution architecture using DASH in CDN. An end-to-end DASH system consists of a server, clients and a content provider.

DASH content preparation prepares the DASH compliant media content in an offline segment creation mode, or in a reactive segment creation. In the offline segment creation mode, the MPD and the media segments for the representations are created before providing client access to the content. In the reactive segment creation mode, the content provider constructs the media segments according to the received HTTP GET request from a media file, such as an MP4 file, which contains multiple representations. In the second mode, for example, the client request may contain the segment start time, segment duration and representation level. The content provider firstly parses the HTTP request and encapsulates part of an MP4 file to a segment as specified in the HTTP request. A DASH content provider sends DASH compliant media data to a DASH server for streaming. The DASH server can be a standard web server to provide dynamic adaptive streaming over HTTP to DASH clients.

The DASH client firstly accesses MPD to obtain the representation and the address to locate each segment for making HTTP GET segment request. The client continuously requests and receives the segments from the DASH server. DASH further enables the client to request media segments from different representations for reacting to varying

network resources. The rate adaptation can occur each time before requesting a segment.

As DASH facilitates the use of the current web replication and caching infrastructure, CDN is used in this paper to deliver media segments to DASH clients. CDN has achieved great success in delivering web and media data. The infrastructure of CDN, such as Akamai, consists of a set of edge servers, which deliver content to clients as shown in Fig. 1. CDN can redirect a client request, such as a GET media segment request, to an appropriate edge server among multiple edge servers.

The edge server selection method, especially the frequencies of switching edge server, affects the performance of serial and parallel segment fetching methods. For locating the edge server from which an end user receives a segment, the DNS request redirection is used to redirect a client request to an appropriate edge server in CDN such as Akamai [20]. The frequencies of switching the edge server depends on the deployed metrics by the DNS based request redirection.

To select an edge server from multiple edge server candidates, the metrics of proximity, content caching possibility and edge server load can be used in a DNS request redirection system [20]. Proximity indicates the distance between the edge server and the client and a close-by edge server will be selected. Edge server load is measured as the bandwidth and processing loads of the edge servers. The edge server load balancing algorithm distributes traffics among edge servers targeting a uniform load distribution at the edge servers. Basically, the proximity based edge server selection method results in the infrequent switching between edge servers since the distance between the edge server and the client is relatively static. In contrast, the load based edge server selection method causes frequent switching when balancing the load between different edge servers in order to improve the overall performance.

To achieve improved load balancing, short time-to-live (TTL) is typically used in the DNS based request redirection, also called DNS resolution. For DNS based request resolution [21], the authoritative DNS name server and local DNS server cooperate in a hierarchical way. The local DNS server sends a DNS request to an authoritative DNS server, which resolves the CDN server name to the IP address of the edge server. In the IP address mapping received from authoritative DNS server, a time-to-live (TTL) is typically contained for the mapping. When the local DNS server queries the authoritative DNS server, the local DNS server will cache the content record for the time specified in the TTL. If the local DNS server receives a query about the caching name server for the mentioned record within the TTL, the local DNS server resolves the edge server IP address with the cached content recoding instead of querying it from the authoritative name server. Krishnamurthy et al. [21] reported that generally a low TTL is assigned in order to assist the DNS based load balancing among the distributed edge servers. The authors claimed that the local DNS servers are forced to process frequent DNS lookups by assigning a small DNS TTL for the content record. Therefore, CDN can obtain control for utilizing the distributed edge servers efficiently by taking advantage of

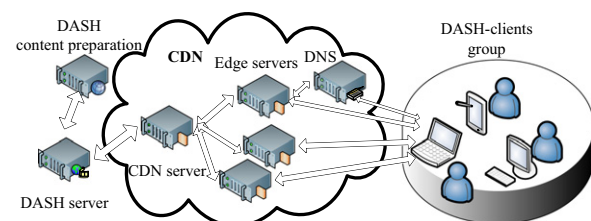


Fig. 1. System for Dynamic Adaptive Streaming over HTTP (DASH) in CDN.

the load balancing algorithm. As an observation of the real CDN request redirection, they reported that the Akamai and Adero, Digital Island, Speedera and Fasttide assigned 10, 20, 20, 120 and 230 s of TTL, respectively.

Recently, Adhikari et al. [22] presented research results of uncovering load balancing and server selection strategies deployed in Youtube using a reverse engineering based methodology. They reported that Youtube deploys the static load sharing using hash based mechanism, a semi-dynamic approach using location aware DNS resolutions and dynamic load-sharing using HTTP redirections. In the first strategy, Youtube maps video-id to a unique hostname in each of the namespace in the hierarchical DNS based host namespaces. In the second strategy, the proximity information is used in request redirection according to the “normal hours” and “busy” hours. In the third strategy, Youtube uses local “intra-tier” load-sharing before resorting to “inter-tier” load-sharing to further balance the load among different edge servers. The research results in [22] showed that the load balancing is deployed in practical applications such as Youtube to smooth the un-even load distribution caused by the combination of spontaneous video demands and dynamic video popularity. The “inter-tier” load-sharing infers that frequent edge server switching is possible to balance the load in the distributed edge servers.

Therefore, redirecting a HTTP request for fetching a specific segment to a specific edge server depends on the load balancing algorithm of DNS based request redirection and TTL value used in CDN. In addition, the dynamic caching status of media segments and the varying network resources further affect the load balancing algorithm to select an appropriate edge server. Several factors, including load balancing algorithm, TTL value used and dynamic caching and network resources together determine the switching between different edge servers for the consecutive requests from the same client. Specifying such frequency is out of the scope of this paper. To focused on rate adaptation algorithm of DASH, the proposed rate adaptation algorithm is presented as transparent of the edge server switching frequencies.

In the content outsourcing, pull-based and push-based outsourcing can be utilized [23]. In the pull-based content outsourcing, the client request is redirected to the selected edge server as discussed according to the above policies using the DNS redirection or URL rewriting methods. The pull-based outsourcing can be classified into uncooperative and cooperative outsourcing. For both uncooperative and cooperative pull-based outsourcing, if the requested segment is cached in the selected edge server, then the client fetches the segment directly from the edge server. In the case of a cache miss, the uncooperative outsourcing redirects the request either to the CDN server such as Akamai server or to the original server. For the cooperative outsourcing, the edge servers cooperate with each other to response to the request from a client, in case of a cache miss. The push-based content outsourcing distributes the contents to the edge servers proactively. As the push-based outsourcing is on the theoretical study stage, this paper focuses on the pull-based outsourcing. For the uncooperative and cooperative outsourcing we focus on the former

strategy as most of CDN providers, such as Akamai, deploy this method [23].

Thanks to the distributed resources offered by CDNs, a DASH service over a CDN can improve the scalability in streaming clients. However, the achievable bandwidth for delivery of media segments is still limited. In media delivery over CDN, congestion might not only occur in the “last mile” of media delivery network, e.g. the link from the Internet service provider to the end user equipment, but also in the distributed edge servers. Furthermore, more and more end users are connected to the Internet using a high-bandwidth access link. Hence, efficient utilization of the distributed edge servers in CDNs is one of the key factors to provide high quality media in DASH. Different segments can be delivered to DASH clients through different edge servers in CDN with request redirection and load balancing algorithm. Therefore, a parallel segment fetching method was proposed in our previous paper [11] to utilize the limited bandwidth in the distributed edge servers more efficiently.

As shown in Fig. 1, the scenario of multiple DASH clients, which compete for the distributed bandwidths in the edge servers, is discussed in this paper. To investigate the uncooperative pull based content outsourcing in the scenario of multiple DASH clients, different DASH clients request a different media clip. By requesting different media clips, each GET segment request is sent to the CDN server, which then pulls the segment from the initial server.

4. Serial and parallel segment fetching methods

In this section, the serial and the parallel segment fetching methods are briefly described, [10,11]. A summary of the terms and a definition of the symbols appearing in this paper are shown in Table 1.

The serial segment fetching method [10] is a common method to request a segment where segments are requested and received one after another. Fig. 2 shows an example of requesting segments serially. The horizontal axis denotes the time of requesting and receiving segments and the period of time to receive segment $#i$ (rec segment $#i$) is depicted in Fig. 2. Note that in general the next segment need not be requested immediately after completing the reception of the previous segment, but rather segment requests can be scheduled according to the buffer occupancy and the rate adaptation strategy of the client. The detailed flowchart of the serial segment fetching method will not be discussed further because of its simplicity.

In the parallel segment fetching method [11] segments are requested in parallel. Fig. 3 shows an example of the parallel segment fetching method. In addition, the time axis is depicted to scale the time period to request and receive segments. As shown in Fig. 3, the parallel segment fetching method maintains a certain number of parallel HTTP threads to request segments in parallel, e.g., two HTTP threads are used in this example. The purple and dark blue bars denote the receiving segments over the first and the second HTTP threads, respectively. The purple and dark blue lines indicate the time axis to request and receive the media segments, respectively. The parallel segment

Table 1
Terms and definition of symbols.

Notation	Definition
A	Slope of the linear function of μ_k used in Eq. (2)
b_{r_c}	Encoded media bitrate of the current representation level r_i
b_{r_i}	Media bitrate of the representation level, wherein r_0 and r_{N-1} denote the lowest and the highest representation
b_{max}	Media bitrate of the highest representation level
b_{min}	Media bitrates of the lowest representation level
bmt_c	Current buffered media time
bmt_{min}	Minimum buffered media time used to determine t_{id}
c	The current representation level
DASH	Dynamic adaptive streaming over HTTP
δ	Ratio of the minimum received portions duration and segment duration to determine τ_0 for smoothing the short-term TCP variation
e_d	Switch-down factor
e_d^c	Switch-down factor calculated by the current representation level
e_d^{min}	Minimum switch-down factor calculated by all representation levels
EPFT	Expected portions fetch time
e_u	Switch-up factor
e_u^c	Switch-up factor calculated by the current representation level
e_u^{max}	Max switch-up factor calculated by all representation levels
ESFT	Expected segment fetch time
φ_k	The ratio of the duration of the received portion and the duration of the full segment, used for scheduling a new parallel HTTP request
ISFT	Ideal segment fetch time
k	Parallel index, wherein the latest segment has the index 0 and the earliest segment has the index $n_p - 1$
μ_k	Threshold of φ_k to schedule a new parallel HTTP request
μ_0	First term of the linear function of μ_k used in Eq. (2)
MPD	Media portion duration denoting the playback duration contained in the received portions
MPD_k	Received media portion duration of a segment with the parallel index k
MSD	Media segment duration denoting the playback duration contained in a segment
MSD_k	Media segment duration contained in the segment with the parallel index k
n_{oo}	Number of the out order received segments
n_{oo}^u	Upper limit of n_{oo}
n_p	Number of parallel HTTP threads to deliver media segments
n_p^u	Upper limit of HTTP threads number to deliver media segments in parallel
PFT	Portion fetch time denoting the spent time for fetching the portions of a segment
PFTM	Portion fetch time based rate adaptation metric
RAM_c	Rate adaptation metric, including PFTM and SFTM, with the rate adaptation index τ
RAM_{τ_0}	The rate adaptation metric with the rate adaptation index τ_0 , which is determined as Eq. (21)
rec segment	spent time to fetch segment #i
#i	
ρ	\widetilde{RSFT}_s factor
r_i	Representation level i
RSFT	Remaining segment fetch time
\widetilde{RSFT}_p	The priority remaining segment fetch time for parallel segment fetching method which will be used in the beginning phase of DASH
\widetilde{RSFT}_s	The priority remaining segment fetch time for serial segment fetching method which will be used in the beginning phase of DASH
SFT	Segment fetch time
SFTM	Segment fetch time based rate adaptation metric
τ	Rate adaptation index of the segment/portions which starts from τ_0 and ends at $\tau_0 + w - 1$
τ_0	The starting rate adaptation index of the segment/chunk
TBMT	Target buffered media time
t_{id}	Idle period of time
t_{req_i}	Time to request a segment #i
ts_i	Playback timestamp of the first frame of a requesting segment # i
ts_{ns}	Playback timestamp of the first frame of the next segment
ts_0	The current playback time stamp at the time instant of requesting the next segment
u_p	Upper limit of parallel HTTP threads
w	Rate adaptation window size which determines the account of RAM used in the rate adaptation

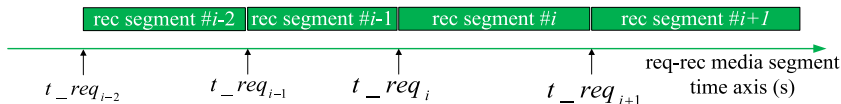


Fig. 2. An example of serial segment fetching method.

fetching method requests segment $i+1$ at $t_{req_{i+1}}$ using the second HTTP thread when segment i is still being received using the first HTTP thread by the client. From

time $t_{req_{i+1}}$ to the time to receive the last byte of segment $\#i$, the two segments, i.e., segment $\#i - 1$ and segment $\#i + 1$ are received in parallel.

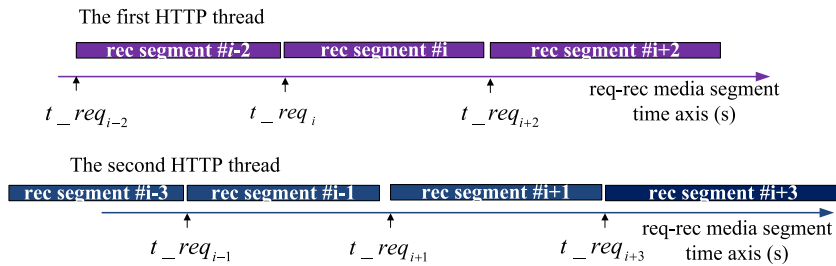


Fig. 3. An example of parallel segment fetching method.

The scheduling method of HTTP GET segment requests in parallel is described as follows. The number of active parallel HTTP threads (n_p) is compared with predefined upper limit (n_p^u) to limit the maximum number of parallel HTTP threads to deliver media segments in parallel. We recommend to set n_p^u as 2, since two parallel HTTP requests already can improve the achievable media bitrates according to the simulation results [11]. In addition, a large value of n_p^u may increase the possibility of receiving segments out of order, which may further cause buffer drain-up, also called buffer underflow.

Each segment received in parallel is identified with a parallel index k , wherein the last requested parallel index is equal to 0 and the earliest requested parallel index is equal to n_p-1 . When requesting a new segment in parallel, all parallel indexes are updated in turn as mentioned above. The parallel index is different from the segment IDs marked as $\#i$ in Fig. 3. In each received segment, the portion of received media data is referred to as received portion and media duration contained in a received portion is denoted as MPD .

The ratio of the duration of the received portion and the duration of the full segment, φ_k , is derived as follows

$$\varphi_k = MPD_k / MSD_k, \quad 0 \leq k \leq n_p - 1 \quad (1)$$

The DASH client compares φ_k with the corresponding threshold μ_k for each parallel received segment k from index 0 to index n_p-1 , to determine if a new HTTP thread needs to start to fetch the next segment. If φ_k is larger than or equal to μ_k for all k from 0 to n_p-1 , then a new request may be sent and the client checks if the remaining conditions to send the next HTTP request are met. Otherwise, (if φ_k is lower than μ_k for any k in the interval $\{0, n_p-1\}$), the new request will not be sent. The remaining conditions to send the next request will be presented in the following paragraphs. To receive the earlier segment in time, a linear function μ_k is used as

$$\mu_k = Ak + \mu_0, \quad 0 < \mu_k < 1 \quad (2)$$

In case μ_k is larger than or equal to 1, it is set as 1. In this paper, A is set as 0.2 as in our previous paper [11]. μ_0 can be set as $1/n_p$. Larger μ_0 and A will assist to successfully receive the earlier requested segments but deduce the efficiency in improving the achievable media bitrates.

In addition, the requested segments may be received out of order. To assist receiving the earlier requested segments in time, the client does not request a new segment in parallel in case the gap between the latest requested segment ID and the earliest requested segment

ID is equal to or larger than its upper limit (n_{oo}^u). The smaller n_{oo}^u will facilitate to receive the earlier requested segments in time but will limit the capability of parallel segment requesting method with respect to improving receiving media bitrates. In this paper, n_{oo}^u is set as 5.

If n_p is smaller than n_p^u , n_{oo} is smaller than n_{oo}^u , and φ_k is larger than or equal to μ_k for all k from 0 to n_p-1 and, a new HTTP request is sent immediately or after a certain idling period derived from the buffered media time. The buffered media time is limited for preventing bandwidth waste due to the fact that a client may stop viewing a certain media clip before playing out the whole downloaded media data. The idling period between two consecutive requests is used to limit the maximum amount of buffered media time. The idling period must not be too long so that the buffered media time can smooth out the most severe bandwidth decrease, i.e., from the highest bandwidth to the lowest bandwidth. Therefore, the idle time period is calculated as

$$t_{id} = bmt_c - bmt_{min} - MSD \cdot b_{max} / b_{min} \quad (3)$$

If t_{id} is greater than 0 then the client idles t_{id} amount of time before requesting the next segment. Otherwise, (if t_{id} is less than or equal to 0), the client requests a new segment immediately. In case of a small $MSD \cdot b_{max} / b_{min}$, bmt_{min} should be set to a relatively large value to increase the condition of bmt_c for idling and to prevent buffer underflow due to sudden bandwidth decrease. In case of a large $MSD \cdot b_{max} / b_{min}$, bmt_{min} can be set as a relatively small value because bmt_c for idling becomes large enough. bmt_{min} is set to 0 in this paper since $MSD(b_{max} / b_{min})$ is already large enough with the selected media bitrates of the representations in the simulation to prevent buffer underflow. For requesting the next segment, the rate adaptation module should be performed to determine the representation level of the next segment. Then, n_p is increased by one when a new HTTP thread is started to request the next segment in parallel.

To conclude the discussion above, the DASH client does not request a new segment and continues to receive the previously requested segments in the following three cases. First, the number of segments being received equals to the limit of the number of parallel HTTP threads, i.e. n_p is equal to n_p^u . Second, φ_k is smaller than μ_k in one of the segments being received. Third, the idle period is greater than 0.

With the chunked transferring mode of HTTP, the DASH server can split a media segment into small portions and transmit a segment as several portions. The client can receive

a segment portion-by-portion until receiving the whole segment. When new portions of a segment are received by the client, the client updates the duration of the received portion (MPD_k) for the parallel received segment. If the whole segment is received by the client, then n_p is decreased by one. Finally, if all segments are received or the user ends the current DASH, then the current parallel segment fetching is terminated. The detailed scheduling algorithm is presented in the following pseudocode.

shortly called as portion in this paper, is received. As the reception data rate of the latest received portion can represent the prevailing network bandwidth, the measured portion fetch time (PFT) is compared with the corresponding expected portion fetch time ($EPFT$), which is used as another rate adaptation metric denoted as PFT rate adaptation metric ($PFTM$).

Specifically, SFT denotes a period of time from the time instant of sending a HTTP GET segment request to the

Scheduling algorithm for HTTP GET segment requests for the parallel segment fetching method in CDN [11]

```

Initialize the current representation level  $i$  as 0
Send HTTP GET request to fetch the first segment immediately
While User has not stopped the current DASH and more segments are available for fetching
  Wait for an event
  If Received portions of the requested segments
    If The received portions of a segment having parallel index  $k$ 
      Update  $MPD_k$ 
    If Received a whole segment
       $n_p = n_p - 1$ 
  Else If Timeout of the scheduled event to send the next HTTP GET request
    Send the next HTTP GET request immediately (after processing the rate adaptation module)
     $n_p = n_p + 1$ 
  If  $n_p < n_p^u$  and  $n_{oo} < n_{oo}^u$ 
     $k = 0$ 
    While  $k < n_p$ 
      If  $\varphi_k \geq \mu_k$ 
         $k = k + 1$ 
      Else
        Break
    End while
  If  $k = n_p$ 
    Calculate the idle period  $t_{id}$  as Eq. (3)
    If  $t_{id} > 0$ 
      Schedule an event to send the next HTTP GET request after idling  $t_{id}$  amount of time
    Else
      Send the next HTTP GET request immediately (after processing the rate adaptation module)
       $n_p = n_p + 1$ 
  End while

```

5. Proposed rate adaptation metric

This section presents the proposed rate adaptation metric for DASH, called segment fetch time (SFT) rate adaptation metric ($SFTM$), which compares $ESFT$ with the measured SFT . The former represents the optimum segment fetch time and the latter denotes the measured segment fetch time. It is well known that the instantaneous TCP transmission rate is dynamically changing hence it is not feasible to measure the streaming media bitrate capacity using the instantaneous HTTP/TCP reception rate. Instead, a DASH client estimates the optimum segment fetch time, i.e., $ESFT$ each time before requesting a segment. After receiving a segment, the estimated $ESFT$ is compared with the measured SFT to determine if the media bitrate of the current representation matches the available end-to-end bandwidth capacity.

In the parallel segment fetching method, the HTTP GET request can be sent when a portion of the previous segment,

instant of receiving the last byte of the requested segment; while the PFT denotes a period of time from the time instant of sending a HTTP GET segment request to the instant of receiving the last byte of the portion of the requested segment. To smooth out short-term variations in HTTP/TCP reception rate, the measuring period should be selected appropriately. Typically a longer period is capable of producing a smoother throughput measurement. However, a longer period will also result in a slower rate adaptation behavior. Based on the theoretical analysis of smoothing short-term TCP throughput reported in our previous paper [24] and experimental results reported in the same paper, a measuring duration around 7–10 s is sufficient to smooth out the varying instantaneous HTTP/TCP reception rates.

5.1. Expected segment fetch time for DASH

The spare network bandwidth and network congestion can be identified by comparing $ESFT$ with the measured segment fetch time. $ESFT$ is determined by the following

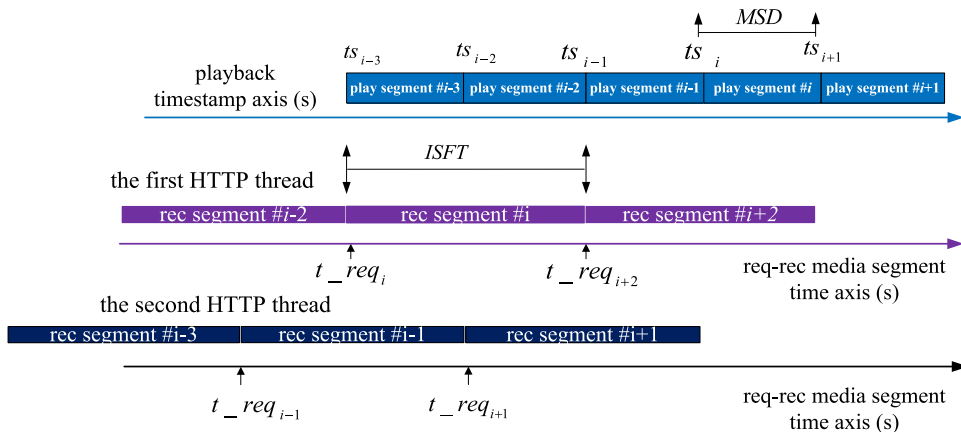


Fig. 4. An example of ideal parallel segment fetching method to demonstrate the relationship between $ISFT$ and MSD .

two factors. One factor is the media segment duration multiplied by the number of parallel HTTP threads, which represents the ideal time to fetch each segment ($ISFT$). The other factor is the available time to fetch the next segment to maintain the target buffered media time ($TBMT$). $TBMT$ is used to smooth the short-term transmission variation. The available time to fetch the next segment is denoted as the remaining segment fetch time ($RSFT$). $ISFT$ represents the intrinsic segment fetch time, which is determined as the period of time to fetch each segment when the media bitrate of each segment exactly matches the capacity of the network resources. The intrinsic segment fetch time relies on the segment fetching method as $ISFT$ is different with the different n_p^u . We will give an example to demonstrate how the segment fetch method affects $ISFT$ after presenting the equation to calculate it. While $RSFT$ represents the real-time buffering status of DASH, $ISFT$, $RSFT$ and $ESFT$ are computed as follows in the following paragraphs.

$ISFT$ is determined as

$$ISFT = MSDn_p^u \quad (4)$$

As discussed in Section 4, the serial segment fetching method can be regarded as a special case of the parallel segment fetching method, wherein n_p^u equals 1.

Fig. 4 shows an example of demonstrating a scenario, in which the media bitrate exactly matches the network capacity. In Fig. 4, the purple and dark blue bars and lines represent parallel HTTP threads as in Fig. 3. In addition, the playback timestamp of the first frame of each segment is labeled as ts_i , the blue bars and line denote playback duration of each segment and playback timestamp axis, which also applies to the following figures depicting the parallel segment fetching method. In the ideal scenario, each segment is fetched using exactly $ISFT$, which equals to the media segment duration multiplied by n_p^u , e.g. 2 in this example. It should be noticed that the ideal scenario does not assume an instantaneous reception rate matching the bandwidth but assumes only an average reception rate matching the bandwidth.

In practice, the above discussed ideal scenario is hard to achieve because of the following reasons. First, there is

typically a slight mismatch between the optimum media bitrates and the network capacity. Second, the network resources and throughput are dynamically changing. To smooth short-term HTTP/TCP throughput variation, it is beneficial to maintain a $TBMT$ indicating a desired latency between the time instant to receive the last byte of a segment to the playback of the first media frame of a segment. The $RSFT$ depends on the current playback timestamp (ts_0) at the time instant of requesting the next segment, the timestamp of the first frame of the next segment (ts_{ns}) and $TBMT$. Hence, $RSFT$ can be calculated as

$$RSFT = ts_{ns} - ts_0 - TBMT \quad (5)$$

Based on Eqs. (4) and (5), finally, $ESFT$ is determined as

$$ESFT = \min(ISFT, RSFT) \quad (6)$$

In this equation, $ESFT$ represents the optimum segment fetch time, which is used in the rate adaptation metrics. $ISFT$ will be constant with a specific n_p^u but $RSFT$ will vary dynamically.

The reason for using $ISFT$ in the proposed rate adaptation metric is to prevent possible fluctuation in representation level. $RSFT$ is another representation of the buffered media time. Hence if only $RSFT$ were used to determine $ESFT$, then the representation level might fail to converge to an optimum representation level as discussed in Section 2. A large $RSFT$ might be a consequence of a cumulative increase of buffered media time. For example, buffered media time may be accumulated due to a step-wise representation-level switch-up strategy. Therefore, $RSFT$ might not provide an optimal value for $ESFT$. In contrast, we propose to determine $ESFT$ as the minimum of $ISFT$ and $RSFT$. The former represents the intrinsic segment fetching time determined by the segment fetching method and the latter represents the real-time buffering status. Hence the proposed method can efficiently facilitate the convergence property but also prevent buffer underflow.

Fig. 5 shows an example of determining $ESFT$ for the parallel segment fetching method. In this example, it is assumed that n_p^u is equal to 2 and $TBMT$ is equal to MSD . The last byte of segment $\#i-2$ is assumed to be received

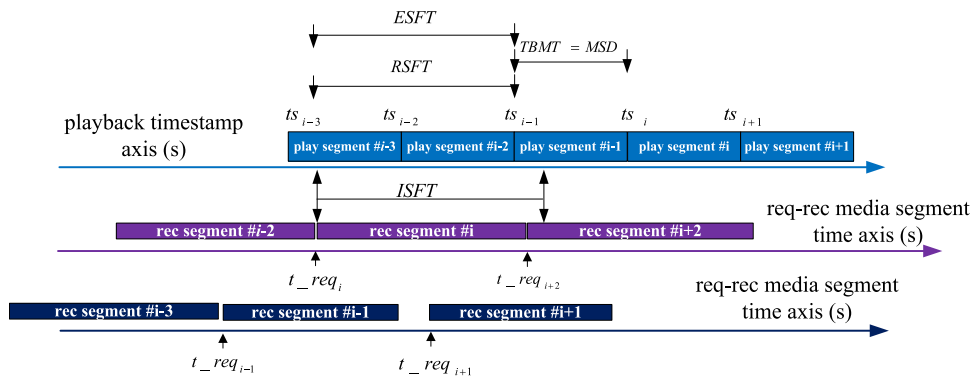


Fig. 5. Example of determining $ESFT$ for the parallel segment fetching method.

at the client later than expected as in the ideal scenario depicted in Fig. 4. Consequently, the HTTP GET request for segment $\#i$ is also sent later than in the ideal scenario. So $RSFT$ is determined using Eq. (5) and is lower than $ISFT$. According to Eq. (6), $ESFT$ equals to $RSFT$ as shown in Fig. 5, which is used in the rate adaptation metrics and the rate adaptation algorithms as discussed in the following sections.

5.2. SFT rate adaptation metric

$ESFT$ represents the optimum duration for fetching a segment. If the measured segment fetch time (SFT) is equal to $ESFT$ calculated before sending the HTTP GET segment request, then each segment is fetched using the optimum segment fetch time and $TBMT$ amount of media data will be maintained at the client buffer. Hence the encoded media bitrate of the current representation matches the network capacity. If SFT is larger than the calculated $ESFT$, then it means that the average TCP throughput for fetching the segment is lower than the bitrate of the current representation level. Otherwise, (if SFT is lower than the calculated $ESFT$), the smoothed TCP throughput is higher than the bitrate of the current representation. The last case can occur in DASH because the TCP sender transmits the available data at the highest possible rate allowed by the TCP congestion control and avoidance algorithm. Hence a ratio of $ESFT$ and the measured SFT is used as the metric denoted as $SFTM$ to detect congestion and to probe the spare network capacity.

$$SFTM = ESFT/SFT \quad (7)$$

where $ESFT$ is calculated as in Eq. (6) each time before sending an HTTP GET request for the next segment and SFT is measured after a segment is fetched. The proposed rate adaptation metric can be used to identify if the media bitrate matches the available network capacity, as our metric measures the reception status in a given duration to smooth out short-term variations caused by the TCP congestion control.

5.3. PFT rate adaptation metric

As PFT measures the time spent to fetch portions of a segment instead of the whole segment, PFT should be compared with the expected portions fetch time ($EPFT$) instead of the $ESFT$, where the $EPFT$ denotes the optimum time to fetch the corresponding portions, which have been received by the DASH client. Therefore, $PFTM$ is computed as follows:

$$PFTM = (ESFT \cdot MPD)/(PFT \cdot MSD) \quad (8)$$

$PFTM$ rate adaptation is useful in the rate adaptation of the parallel segment fetching method. In the parallel segment fetching method, an HTTP GET request can be sent when the previously requested segments have not been completely received at the DASH client. In such a scenario, PFT represents the latest reception status and reveals the current network situation. On the other hand, the measuring duration should be larger than a certain duration to smooth out short-term TCP throughput variations. Several methods of partitioning a segment into sub-segments dynamically and requesting parallel sub-segments, instead of requesting parallel segments, were discussed in [19]. In this paper, we focus on the rate adaptation and PFT used in one of the rate adaptation metrics.

One advantage of $SFTM$ and $PFTM$ is that our methods do not require information from the transport layer (TCP layer). Our method uses the reception status at the application layer, but does not rely on the TCP throughput calculation models. In order to use the TCP throughput calculation equations, the packet loss rates, RTT or the current congestion window are required; however, such information is not available at the application layer. In addition, if the current congestion window is included in the TCP throughput calculation equation, then the TCP throughput equation basically represents the instantaneous transmission rate instead of the smoothed TCP throughput. As it is well known, the instantaneous TCP throughput is varying and cannot be directly used in the rate adaptation to estimate the end-to-end network capacity. In contrast, our rate adaptation metrics can smooth out the varying TCP throughput rate by measuring a certain duration of the reception status. Therefore,

our rate adaptation metrics can be applied for the rate adaptation of DASH, which is operated in the application layer.

6. Proposed rate adaptation algorithm for the serial segment fetching method

In this section, we propose a rate adaptation algorithm for the serial segment fetching method based on *SFTM*, presented in Section 5. In the serial segment fetching method, the HTTP GET next segment request is sent after receiving the current segment. The rate adaptation algorithm occurs each time after receiving a media segment and before requesting the next segment to determine the representation level of the next segment.

In the serial segment fetching method, since *ISFT* is equal to *MSD*, Eq. (7) can be expressed according to Eq. (6) as

$$SFTM = \text{MIN}(MSD, RSFT) / SFT \quad (9)$$

In our previous work [10], a simplified rate adaptation metric was presented, which was set to the ratio of *MSD* and *SFT*. The rate adaptation metric in [10] only considers *ISFT* of the serial segment fetching method but does not take into account *RSFT*.

In real-time DASH, different *RSFT*s of different DASH clients will cause unfairness especially to those short lived and lately joined clients. In the beginning of DASH, if *RSFT* calculated by Eq. (5) is smaller than the priority *RSFT* for a serial segment fetching method (\overline{RSFT}_s) in Eq. (10), then *RSFT* is replaced by \overline{RSFT}_s . Once *RSFT* calculated by Eq. (5) is larger than \overline{RSFT}_s , *RSFT* will always be set as Eq. (5).

$$\overline{RSFT}_s = \rho \cdot MSD \quad (10)$$

In this paper, ρ is selected as 0.75. \overline{RSFT}_s is especially important for DASH clients whose initial buffered media time is much lower than the *ISFT*. The priority *RSFT* has a larger impact on the parallel segment fetch method compared to the serial segment fetching method since the former occupies the larger *ISFT* compared to the later. The priority *RSFT* will be further discussed in Section 7.

The rate adaptation deploys a step-wise switch-up and a multi-step based switch-down method to change the representation level. The condition of switch-up/down and the corresponding operations are described as follows:

Switch-up: It takes place if the following condition holds:

$$SFTM > 1 + \varepsilon_u \quad (11)$$

In Eq. (11), the left term represents the metric to detect spare network bandwidth and the right term denotes the condition to switch-up to the next higher representation level. ε_u is determined according to the maximum bitrates jump-up ratio of all representation levels ε_u^{max} and bitrates jump-up ratio of the current representation level (ε_u^c). ε_u^{max} is calculated as

$$\varepsilon_u^{max} = \max\{(b_{r_{i+1}} - b_{r_i}) / b_{r_i}, \quad \forall i = [0, 1, \dots, N-1]\} \quad (12)$$

ε_u^{max} provides a conservative switch-up factor, which improves the convergence property of the proposed rate

adaptation algorithm. And ε_u^c is calculated as

$$\varepsilon_u^c = (b_{r_{c+1}} - b_{r_c}) / b_{r_c} \quad (13)$$

ε_u^c provides a more aggressive switch-up factor compared to ε_u^{max} . Since ε_u^c is only determined by the bitrate jump ratio of the current representation level and ε_u^{max} is decided by the maximum of bitrate jump ratios of all representation levels. To reach a compromise between the convergence of the rate adaptation algorithm and the achievable media bitrates, ε_u is calculated as

$$\varepsilon_u = \min\{\varepsilon_u^{max}, 2\varepsilon_u^c\} \quad (14)$$

Eq. (14) not only improves the convergence and achievable media bitrate, but also makes the rate adaptation algorithm performance independent of the bitrate distribution of the representation levels. For example, if ε_u^{max} is much larger than ε_u^c , then ε_u can be determined as $2\varepsilon_u^c$, which enables the rate adaptation algorithm to switch-up the representation level to the optimum level. Otherwise, (ε_u^{max} is smaller than $2\varepsilon_u^c$), ε_u will be decided by ε_u^{max} for improving the convergence.

In the case of a decision to switch-up, the rate adaptation algorithm selects the next higher representation level. The reason for using a conservative step-wise switch-up strategy is to prevent playback interruption which might occur in case of aggressive switch-up operations. During a step-wise switch-up, moreover, buffered media time can rise to a more safety level to prevent buffer underflow due to a sudden bandwidth decrease. So the initial buffering time, spent to buffer the media time to reach its minimum, can be set as a relatively small value to reduce the delay before starting playback.

Switch-down: It will be performed if the following inequality holds

$$SFTM < 1 - \varepsilon_d \quad (15)$$

The left term of Eq. (15) compares the selected media bitrate to the current bandwidth in terms of ratio of the optimum segment fetch time to the actual segment fetch time, while the right term represents the condition to switch-down to a lower representation level.

To identify ε_d , the minimum bitrate switch-down ratio of all representation levels (ε_d^{min}) and bitrates switch-down ratio of the current representation level (ε_d^c) are used. ε_d^{min} is calculated as

$$\varepsilon_d^{min} = \min\{(b_{r_i} - b_{r_{i-1}}) / b_{r_i}, \quad \forall i = [0, 1, \dots, N-1]\} \quad (16)$$

ε_d^c is calculated as

$$\varepsilon_d^c = (b_{r_c} - b_{r_{c-1}}) / b_{r_c} \quad (17)$$

Then ε_d is calculated as

$$\varepsilon_d = \max\{2\varepsilon_d^{min}, \varepsilon_d^c\} \quad (18)$$

In case of congestion, *SFT* will be much larger than *ESFT*, i.e. RAM is lower than 1. Hence Eq. (18) selects a switch down factor at least larger than a specific value such as $2\varepsilon_d^{min}$ which is used in this paper. When ε_d^c is larger than $2\varepsilon_d^{min}$, we set ε_d as ε_d^c . Eq. (18) enables the proposed rate adaptation algorithm to provide an advanced convergence property. For a rate adaptation algorithm of DASH, providing an advanced convergence property is one of most challenging task since the switch-up and switch-down behaviors from different DASH

clients will affect other clients. In this paper, we propose Eqs. (14) and (18) to provide a conservative switch-up and switch-down for improving the convergence property.

The proposed rate adaptation algorithm of the serial segment fetching method is described as the following pseudocode.

Proposed Algorithm 1. Rate adaptation of the serial segment fetching method operated at the client.

```

Initialize the current representation level  $i$  as 0
Send HTTP GET the first segment request immediately
While User has not stopped the current DASH and more segments are available for fetching
  Wait for an event
  If (Received a segment and  $t_{id} \leq 0$ ) or (timeout to request the next segment)
    If switch up condition meets
      Set representation level to the next higher level
    Else if switch down condition meets
      Representation level  $i$  is determined as the first representation  $r_i$  to meet the inequality (19)
    Else
      Representation level remains unchanged
      Send HTTP GET the next segment request immediately
  If Received a segment and  $t_{id} > 0$ 
    Idle  $t_{id}$  period before sending HTTP GET the next segment request
End while

```

To support the conservative switch-down, a client buffers enough amount of media time to recover from a severe bandwidth decrease. To accumulate media time, this paper deploys an additive increase and multiplicative decrease strategy to adapt the representation level.

In the switch-down, an aggressive switch-down will be performed. The selected representation level is determined to be the first representation (in descending order) with level r_i to meet

$$b_{r_i} < SFTM \cdot b_{r_c} \quad (19)$$

The idle time calculation algorithm is deployed before sending the next GET request in order to prevent client buffer overflow, in a similar manner as Eq. (3) presented in the parallel segment fetching method.

7. Proposed rate adaptation algorithm for parallel segment fetching method

In this section, we present a novel rate adaptation algorithm for the parallel segment fetching method based on the proposed rate adaptation metrics, i.e. *SFTM* and *PFTM*. In this section, *SFTM* and *PFTM* are uniformly denoted as rate adaptation metric *RAM*.

The rate adaptation algorithm also deploys a step-wise switch-up and an aggressive switch-down method, which are determined according to the following two cases.

Switch-up: It takes place if all RAM_τ of the parallel received segments meet the following condition

$$RAM_\tau > 1 + \varepsilon_u, \forall \tau = [\tau_0, \tau_0 + 1, \dots, \tau_0 + W - 1] \quad (20)$$

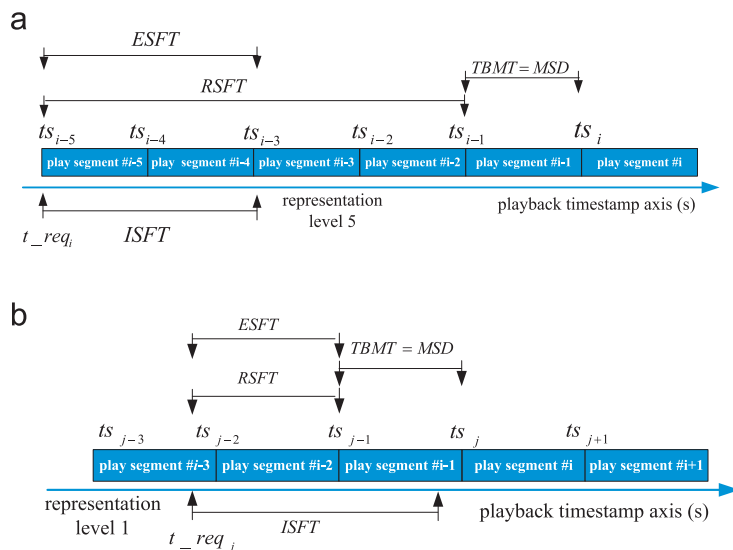


Fig. 6. An example of demonstrating unfairness between different clients due to the different *RSFTs*: (a) first client playback timestamp of each segment and the *ESFT* to fetch the segment # i and (b) second client playback timestamp of each segment and the *ESFT* to fetch the segment # j .

where the left term represents the metric to detect congestion and the right term denotes the condition to switch up to the next higher representation level. In the above switch-up condition, a sliding window is used to compare the latest w number of RAM_τ with the switch-up threshold to decide whether or not to perform a switch-up. And the rate adaptation index (τ) starts from τ_0 instead of 0 to set a minimum received portion media duration for each RAM_τ . RAM_τ can be used as the rate adaptation metric if the ratio of the received portion duration and the segment duration is larger than a predefined threshold (δ) or PFT is larger than $ISFT$ as

$$\tau_0 = \arg_{\min(k)} \left[\frac{MPD_k}{MSD} \geq \delta \text{ or } MPD_k > ISFT \right],$$

$$\forall k = [0, 1, \dots, n_p - 1] \quad (21)$$

The reason of setting τ_0 is to smooth the short-term TCP transmission variation. δ is set as 0.7 in this paper. ε_u is calculated as in Eq. (14).

Switch-down: It will be performed if the RAM_{τ_0} meets the following condition:

$$RAM_{\tau_0} < 1 - \varepsilon_d \quad (22)$$

τ_0 is the starting rate adaptation index determined as Eq. (21). ε_d is calculated as in Eq. (18), which provides a conservative switch-down performance since ε_d is set as at least larger than $2\varepsilon_d^{\min}$.

In the switch-down, an aggressive switch-down will be performed. The selected representation level is determined to be the first representation (in descending order) with level r_i to meet

$$b_{r_i} < RAM_{\tau_0} b_{r_c} \quad (23)$$

The fairness between different clients is one of the important factors to evaluate the efficiency of a rate adaptation algorithm. Fig. 6. shows an example demonstrating the occurrence of an unfairness between the two clients because of the sustained different $ESFT$ s. Fig. 6(a) and (b) depict the $ESFT$ s of the first and the second clients, respectively. In Fig. 6, the same term and definition of the symbols are used as in Fig. 4 to depict the playback time-stamps of the first frame of each segment, the playback durations of each segment and the playback timestamp axes. It is assumed that the first client starts DASH earlier than the second client. In this example, we assume that the spare network bandwidth is available when the first client starts DASH and the network bandwidth is scarce when the second client starts DASH. In Fig. 6(a), the first client reaches a relatively higher representation level 5 and buffers a large amount of media time at time instant t_{req_i} using the spare networks. The second client in Fig. 6 (b), however, stays at a lower representation level 1 and only buffers a small amount of media time at time instant t_{req_j} as the networks enters a saturated status when the second client starts DASH. The different buffered media time results in different $RSFT$ s, which further causes different $ESFT$ s as demonstrated in Fig. 6.

To the first client, shown in Fig. 6(a), the current playing segment is segment $i-5$ when the first client requests segment i at time instant t_{req_i} . In this example, $ISFT$ equals to two times MSD as n_p^i is set as 2 and $TBMT$ equals MSD .

Since $RSFT$ covers more than two segment durations, it is larger than $ISFT$. So $ESFT$ needed to fetch segment i is equal to $ISFT$ according to Eq. (6). For the second client, shown in Fig. 6(b), the current playing segment is $j-3$ when the second client requests segment j at time instant t_{req_j} . Under the same $TBMT$ with the first client, $RSFT$ of the second client is less than $ISFT$. Thus $ESFT$ required to fetch segment j is equal to $RSFT$ according to Eq. (6) for the second client. As networks undergo a saturated status, it might be difficult for the second client to accumulate buffered media time and $RSFT$ may be sustainably lower than $ISFT$. In real-time DASH, different $ESFT$ s of different clients will cause different achievable media bitrates, since $ESFT$ determines RAM , $SFTM$ and $PFTM$, according to Eqs. (7) and (8).

To solve the problem of unfairness caused by different $RSFT$ s of different clients, this paper presents a method of deploying the priority $RSFT$ for parallel segment fetching method (\overline{RSFT}_p) in the beginning of DASH. If $RSFT$ calculated by Eq. (5) is smaller than \overline{RSFT}_p calculated by Eq. (24), then $RSFT$ is replaced by \overline{RSFT}_p . Once $RSFT$ calculated by Eq. (5) is larger than \overline{RSFT}_p , $RSFT$ will be calculated as in Eq. (5).

$$\overline{RSFT}_p = (1 + (n_p^u - 1)/2)MSD \quad (24)$$

\overline{RSFT}_p allows newly started DASH clients to successfully switch-up to the appropriate medium representation level.

The rate adaptation algorithm of the parallel segment fetching method is shown in the following pseudocode, which operates as the rate adaptation module in Algorithm 1.

Proposed Algorithm 2. Rate adaptation of the parallel segment fetching method operated at DASH client.

```

If switch-up condition is met
  Set representation level to the next higher level
Else if switch-down condition is met
  Representation level  $i$  is determined with the first
  representation  $r_i$  to meet Eq. (23)
Else
  Representation level remains unchanged
End

```

8. Simulation results

The proposed rate adaptation algorithms for the serial and the parallel segment fetching methods in CDN are implemented in ns2 [25]. In ns2, the proxy cache module consists of the standard proxy cache, the client and the server which operate on HTTP/TCP protocols. Recently, Müller and Timmerer [26] presented a reference software for DASH so that rate adaptation algorithms can be implemented in the reference software for evaluation. The bandwidth dynamic and packets loss are emulated using the traffic shaping tools operated on the router or at the end user device. However, emulating the distributed networks such as CDN is not an easy task. For emulating CDN, first, a distributed network topology should be created. For evaluating the dynamics of the distributed

network resources, each edge server should be equipped with the traffic shape tool. Second, it is complex to specify the transporting pass for traffics in the distributed networks, e.g., determining the route between the DASH server and the DASH client for each segment. In contrast, all of the above issues are easy in ns2. Therefore, ns2 is a good choice for evaluating the rate adaptation method for DASH in CDN.

In this paper we evaluate the rate adaptation algorithm in the aspects of (a) convergence in the requesting representation level indicating the media bitrates, (b) fairness between different clients, which compete for the edge server bandwidth, (c) achievable media bitrate, (d) rate adaptation speed and (e) buffer underflow frequency.

To our knowledge, few rate adaptation algorithms are published in the literature. Several HTTP streaming players exist such as Microsoft smooth streaming [5] and Adobe OSMF player [6] have their own internal rate adaptation techniques; however, they are not public. Hence it is not feasible to compare the proposed rate adaptation algorithm with those players. Although we cannot implement the specific rate adaptation algorithm, it is possible to observe the behavior of the commercial players. As mentioned in the related work Section of this paper, Akhshabi et al. [15] reported the experimental results and the evaluation focusing on the evolution of requesting media bitrates over time using commercial players. In Section 8.5 of this paper, samples of the representation level evolution are reported for 10 different clients that compete for the bandwidth of the edge servers. The proposed rate adaptation algorithms are compared with the Smooth streaming reported in [15].

Reporting samples of evolution of the representation levels has limitation in the coverage of reporting results considering the page limitation. For example, when the

DASH client group size was increased to 10, 20 and 30 or even more, it is not feasible to observe each client evolution process. To conduct and report extensive simulation results, the distribution of the representation levels of DASH clients are reported. The histogram of the representations level, which indicates the occurrence frequency of each representation level, is an efficient tool to evaluate the distribution of the representation levels. Reporting each client histogram will also take many pages with the large number of simulated clients. Hence, each histogram is represented as the combination of two values, which capture the features of the histogram and the comprehensive results and details are reported in Section 8.2. The mean of the average media bitrates is reported to evaluate the achievable media bitrates of the overall clients and the buffer underflow frequency is reported, in Sections 8.3 and 8.4, respectively. Finally, samples of the representation level evolutions are reported in Section 8.5.

Table 2
Two sets of bandwidth settings.

Bandwidth settings	Links #1	Links #2	Links #3
Bandwidth setting #1 (Mbits/s)	5.0	2.5	2.5
Bandwidth setting #2 (Mbits/s)	9.0	6.0	6.0

Table 3
Group size with the bandwidth setting #1.

Bandwidth settings	Minimum	Maximum	Step
Bandwidth setting #1	8	32	8

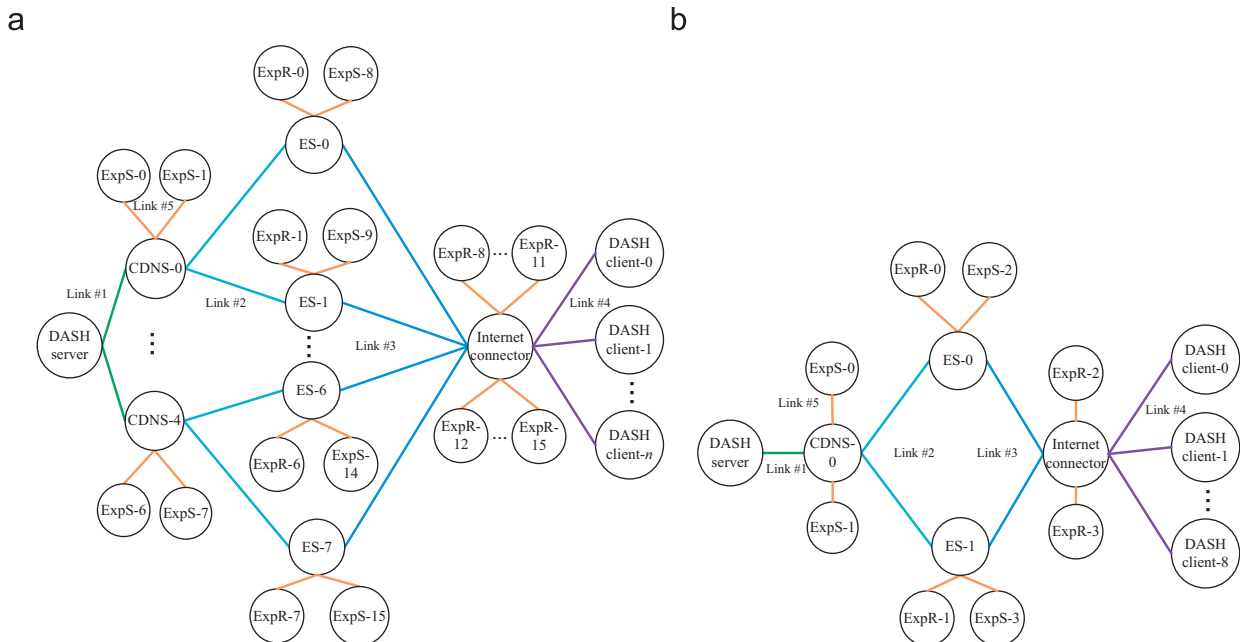


Fig. 7. Network topology of the simulation. (a) Topology #1, (b) Topology #2.

8.1. Simulation topology and setting

Fig. 7 shows the network topologies used in the simulation. In the first topology Fig. 7(a), two layers of media distribution networks are deployed including four CDN servers (CDNS-0, CDNS-1, CDNS-2 and CDNS-3) and eight edge servers (ES-0, ES-1, ..., ES-7). In topology #2, two layers of media distribution networks are deployed including one CDN server (CDNS-0) and two edge servers (ES-0 and ES-1). As shown in Fig. 7, the media segments are delivered from HTTP streaming server to CDN servers, edge servers, the Internet connector and finally to DASH clients. The edge server fetches the segment from the connected upper layer CDN server, which then fetches the segment from the DASH server. The DASH server contains all representations of the media clip. In this paper, all segments are fetched from the DASH server. Caching scenario is not considered in this paper. To simulate the above scenario, different clients request the different media clips, hence all segments have not been cached by the edge servers when they receive a HTTP GET segment request.

Table 2 shows the bandwidth settings of links #1, links #2 and links #3 in Fig. 7. The bandwidths in links #4 and links #5 are set as 2.0 Mbits/s. All link delays are set to 2 ms. Topology #1 in Fig. 7(a) and the bandwidth setting #1 are used to report the simulation results of Sections 8.2, 8.3 and 8.4. Topology #2 in Fig. 7(b) and bandwidth setting #2 are used to report the representation level evolution in Section 8.5.

To simulate practical congestion in real networks and evaluate the rate adaptation algorithm, multiple DASH clients are simulated, which compete for the bandwidths of the distributed edge servers in CDN. The clients are classified into two groups. For the first group, clients start the DASH randomly in the time interval between 0 s and 10 s and end at 1200 s. For the second group, clients start the DASH randomly in the time interval 400–410 s and stop at 800 s. By increasing or reducing the number of clients, denoted as group size, the rate adaptation efficiency can be evaluated for different levels of congestion. Here the group size is the sum of the client's number in the two groups.

Table 3 shows the minimum group size, maximum group size and group size jump step for the bandwidth setting #1. As shown in Table 3, the group size is much larger than the number of edge servers. Hence multiple clients will compete for the bandwidth of the links between the CDN server and the edge server, and the links between the edge server and the Internet connector in Fig. 7. For the bandwidth setting #1, the simulation runs multiple times with different group sizes. The numbers of clients in the first and the second group are equal.

For the first DASH client group, the simulation time is divided into three periods to report simulation results. The three periods include 0–400 s, i.e., before starting the second client group, 400–800 s, i.e., when the second client group compete for the bandwidth of the distributed networks, and 800–1200 s, i.e., after ending the second client group. For the second client group, the simulation results are reported in the period starting from 400 s and ending at 800 s.

To simulate the dynamics of the Internet, background traffic is added to each link between the CDN server and the edge server, and between the edge server and the Internet connector by attaching the exponential traffic senders (ExpS-id) and the corresponding receivers (ExpR-id). Specifically, the exponential traffic senders are attached to the CDN servers (CDNS-id) and edge server (ES-id). The corresponding exponential receivers are attached to the edge servers (ES-id) and the Internet connector. Both operating and idling periods of the exponential traffic are set to 0.5 and the average bitrate of exponential traffics during the operating period is set to 0.2 Mbits/s. The exponential traffics start from 0 s and end at 1200 s.

The edge server selection method and the frequency of switching the edge server affect the achievable media bitrates of the serial and the parallel segment fetching methods. When a simple edge server selection method such as round-robin strategy, is used, the frequent switching of the edge server balances the edge server loads better than the infrequent switching and will more efficiently utilizes the bandwidths of distributed edge servers. Hence the achievable media bitrate of serial and parallel segment fetching methods can be improved. As discussed in Section 3, the proximity based edge server selection technique causes infrequent request redirection, and the short TTL based load balancing algorithm results in frequent edge server switching. Recall that the edge server selection method and determination of the frequencies of switching edge server are out of the scope of this paper. Therefore, the different frequencies of the switching edge server are simulated in order to evaluate the proposed rate adaptation algorithms for the serial and the parallel segment fetching methods with respect to the different switching frequencies. In the simulation, the frequencies of 1 and 0.25 are used for switching edge servers, indicating switching edge servers every one segments and four segments, respectively. In addition, clients randomly select the initial edge server from the set of edge servers to fetch the first segment. Under a specific switching frequency, round-robin strategy is used to switch the edge server.

In our simulations, *TBMT* and the initial buffered media time are set to 20 s. For achieving adaptive HTTP streaming, the server provides ten sets of representations for clients to adapt the media bitrates wherein the media bitrates include 64, 128, 192, 256, 384, 512, 640, 896, 1152 and 1408 Kbits/s and the corresponding representation level starts from 0 and ends at 9, respectively. Bitrate changing step was set to increase as representation level reaches to a higher level in order to achieve an equivalent quality improvement each time bitrate switches-up.

8.2. Distribution of representation level

The histogram of the representation level r_i is an efficient metric for evaluating the rate adaptation algorithm since the histogram indicates the frequency distribution of the inputs. In the histogram calculation, the input is each segment representation level $x = \{x_0, x_1, \dots, x_{H-1}\}$ and the output includes a vector of the occurrence frequency of all possible representation levels

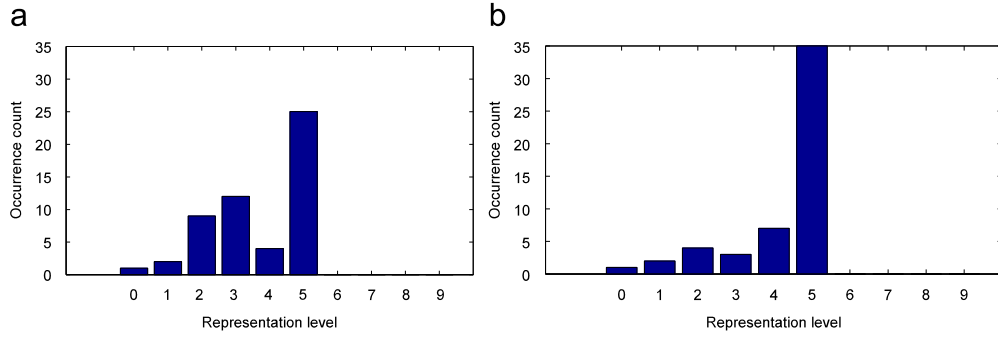


Fig. 8. An example of representation level histograms of two clients and corresponding (σ_f^2, σ_l^2) : (a) first client histogram (65.57, 2.33) and (b) second client histogram (114.84, 0.5).

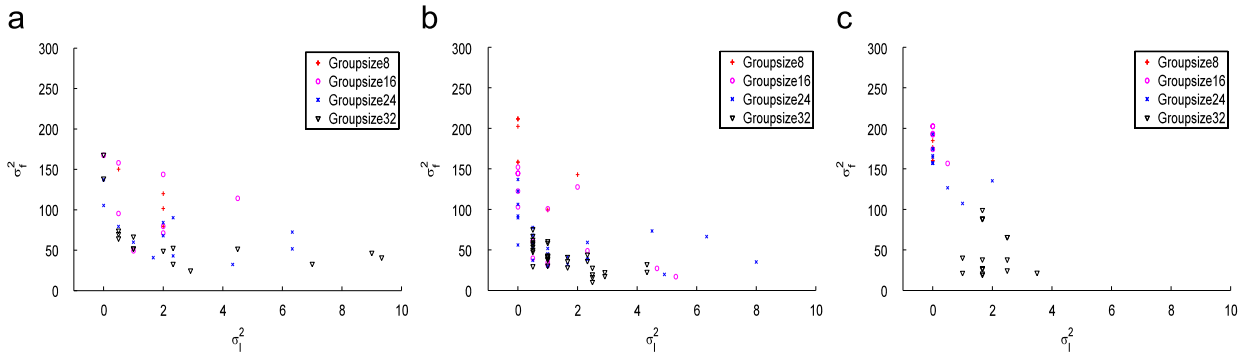


Fig. 9. Representation level distribution of each client using the proposed rate adaptation algorithm for the serial segment fetching method under the edge server switching frequency 1: (a) 0–400 s, (b) 400–800 s and (c) 800–1200 s.

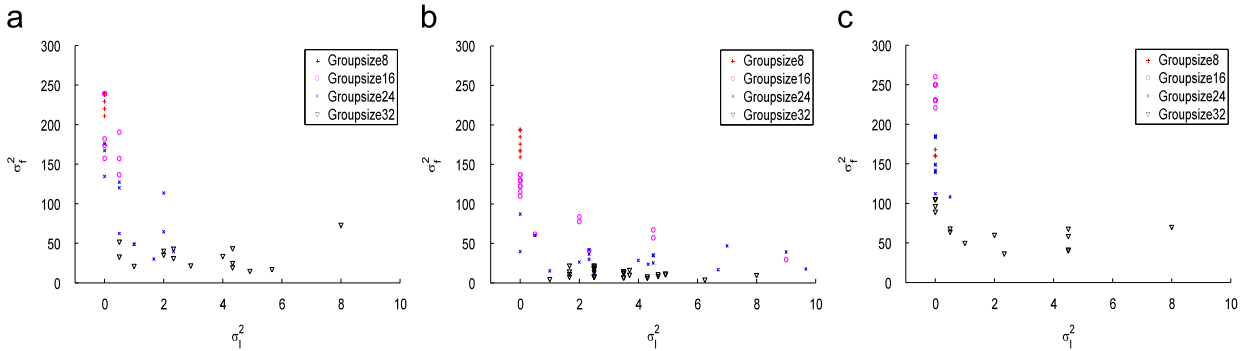


Fig. 10. Representation level distribution of each client using the proposed rate adaptation algorithm for the parallel segment fetching method under the edge server switching frequency 1: (a) 0–400 s, (b) 400–800 s and (c) 800–1200 s.

$c_r = \{c_{r_0}, c_{r_1}, \dots, c_{r_{M-1}}\}$ and a vector of the distinct representation level location $\hat{r} = \{\hat{r}_0, \hat{r}_1, \dots, \hat{r}_{N-1}\}$. In the former vector, the vector size of occurrence frequency M is equal to the number of all possible representation levels provided by the content provider and the occurrence count is set as zero for a representation level that has not occurred. In the later vector, if a representation occurrence count is larger than or equal to a constant threshold θ , then the corresponding representation level is regarded as a distinct representation level and the corresponding representation level value denotes the distinct representation level location. In this

paper, θ is set as 5. Such a distinct representation level locations (\hat{r}_k) forms a vector of distinct representation level locations denoted as \hat{r} .

$$\hat{r} = \{\hat{r}_0, \hat{r}_1, \dots, \hat{r}_{N-1}\} = \arg_{r_k} (c_{r_k} \geq \theta, k = 0, 1, \dots, M-1) \quad (25)$$

where N denotes the size of distinct representation level location, which is smaller than or equal to M .

The variance is calculated over c_r and \hat{r} to obtain the variance of the occurrence frequency of all possible representation levels (σ_f^2) and the variance of distinct

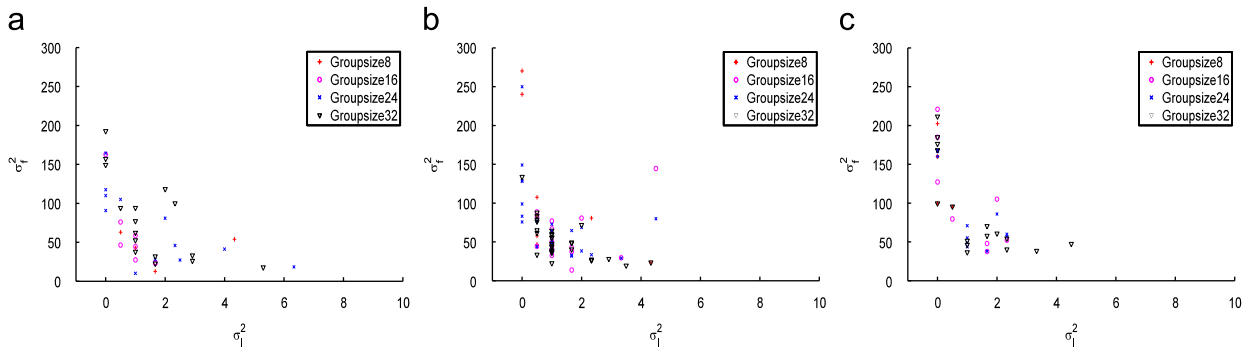


Fig. 11. Representation level distribution of each client using the proposed rate adaptation algorithm for the serial segment fetching method under the edge server switching frequency 0.25: (a) 0–400 s, (b) 400–800 s and (c) 800–1200 s.

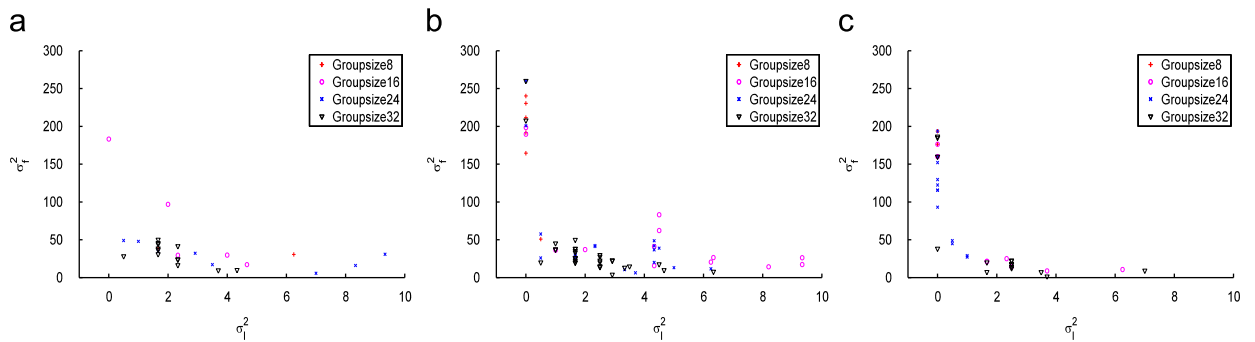


Fig. 12. Representation level distribution of each client based on the proposed rate adaptation algorithm for the parallel segment fetching method under the edge server switching frequency 0.25: (a) 0–400 s, (b) 400–800 s and (c) 800–1200 s.

representation level location (σ_r^2) as follows:

$$\sigma_f^2 = \frac{\sum_{i=0}^{M-1} (c_{r_i} - \bar{c}_r)^2}{M} \quad (26)$$

$$\sigma_l^2 = \frac{\sum_{i=0}^{N-1} (\hat{r}_i - \bar{\hat{r}})^2}{N} \quad (27)$$

where \bar{c}_r denotes the mean of the occurrence frequency of all possible representation levels and $\bar{\hat{r}}$ denotes the mean of distinct representation level locations, respectively. A large σ_f^2 and a smaller σ_l^2 indicate a superior convergence property and vice versa.

Fig. 8 shows an example of calculating σ_f^2 and σ_l^2 for two given histograms of representation levels and comparing the convergence property of the two histograms by comparing the combination of σ_f^2 and σ_l^2 , denoted as (σ_f^2, σ_l^2) , of the two histograms. As shown in Fig. 8, c_r and \hat{r} of the first and the second histogram are $\{1, 2, 9, 12, 4, 25, 0, 0, 0, 0\}$, $\{2, 3, 5\}$, and $\{1, 2, 4, 3, 7, 35, 0, 0, 0, 0\}$, $\{4, 5\}$, respectively. As can be observed from Fig. 8, the second histogram represents a superior convergence compared to the first histogram because of the following two reasons. First, the shape of the second histogram is sharper than that of the first histogram. Second, the distinct representation levels are more focused in the second histogram compared to the first histogram. According to Eqs. (26) and (27), (σ_f^2, σ_l^2) of the first and the second histogram are $(65.77, 2.33)$ and $(114.84, 0.5)$, respectively. The second histogram gives a larger σ_f^2 and a smaller σ_l^2 compared to the first histogram. As mentioned before a

larger σ_f^2 and a smaller σ_l^2 indicate a superior convergence, hence the second histogram represents a more advanced convergence, which is consistent with our observation.

Figs. 9 and 10 plot the (σ_f^2, σ_l^2) of each client for the rate adaptation algorithms of the serial and the parallel segment fetching methods, respectively, in the interval of 0–400 s, 400–800 s, 800–1200 s, with the edge server switching frequency 1. Figs. 11 and 12 show the representation level distributions, similar to Figs. 9 and 10 but with the edge server switching frequency 0.25. The horizontal and vertical axes represent σ_r^2 and σ_f^2 , respectively. The dots distributed in the left-up corner of the figure indicate better convergence compared to the dots distributed in the right-bottom corner of the figure according to the above mentioned rule that a larger σ_f^2 and a smaller σ_l^2 indicate better convergence performance. Four sets of group sizes from group size 8 to 32 are reported.

Comparing the convergence properties of the three periods in Figs. 9 and 10, the best results are given in the period of 800–1200 s, medium results are in period 0–400 s and the worst results are shown in period 400–800 s for the proposed rate adaptation algorithms. The reasons are as following. In the period of 800–1200 s, less clients compete for the bandwidth compared to the period of 400–800 s and the starting representation level is larger than 0. In the period of 0–400 s, the representation level starts from 0. In the period of 400–800 s, more clients compete for the bandwidth compared to the other periods. In addition, the convergence property of the rate

adaptation algorithms depends on the group size according to the simulation results in Figs. 9 and 10. When the group size is 8 or 16, the rate adaptation algorithms provide an improved convergence. The convergence level is still acceptable for the group size 24. But it becomes worse when the group size was increased to 32, which needs to be improved further. However, the above results already show that the proposed rate adaptation algorithm is superior to the behavior of the Smooth streaming player, wherein the two players compete for the bandwidth of a common link, which was reported in section 6 of [15]. According to the evaluation reported in [15], the second player, which started later than the first player, failed to switch-up to the appropriate level and kept oscillating between some lower representation levels due to the competition from only two players. Improved convergence is reported in Figs. 9 and 10 with the group size from 8 to 24 compared to the results and evaluation in [15], wherein the group size was 2.

Providing improved convergence for a large group of DASH clients in CDN is still an open and challenging task because of the following reasons. In the scenario of multiple clients competing for the bandwidth in the

distributed edge servers, the number of clients may be dynamically changing, which causes variation in the sharable bandwidth. Furthermore, the rate adaptation of one client might cause a change in the rate adaptation of other clients because of the varying sharable bandwidth. For example, when several clients compete for the bandwidth in a common link, a switch-up behavior of a client may cause the reduced sharable bandwidth for other clients, while, a switch-down behavior of a client may result in the increased sharable bandwidth in the other clients. Such a relation between different clients is called inter-client effect in this paper. The inter-clients affection will increase along with an increase in the group size. Moreover, the edge server selection and load balancing algorithm affect the variation in the sharable bandwidth of each client and hence affect the convergence property of the rate adaptation algorithm of DASH.

Comparing the rate adaptation algorithms for the serial and parallel segment fetching methods, shown in Figs. 9, 11, Figs. 10, 12, respectively, the simulation results show that the rate adaptation algorithm for the serial segment fetching method slightly outperforms the rate adaptation algorithm for the parallel segment fetching

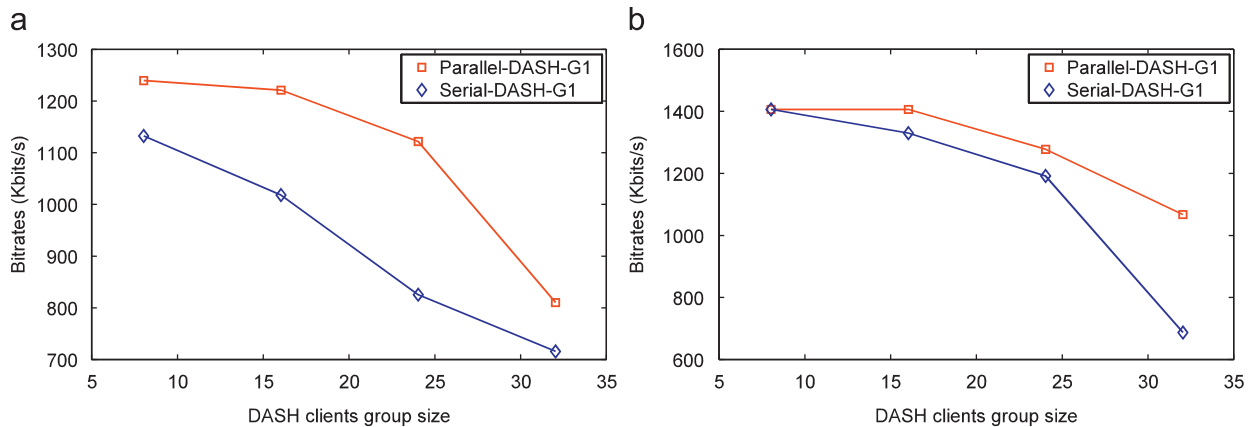


Fig. 13. Mean of average bitrates of the first group with edge server switching frequency 1: (a) the first group in 0–400 s and (b) the first group in 800–1200 s.

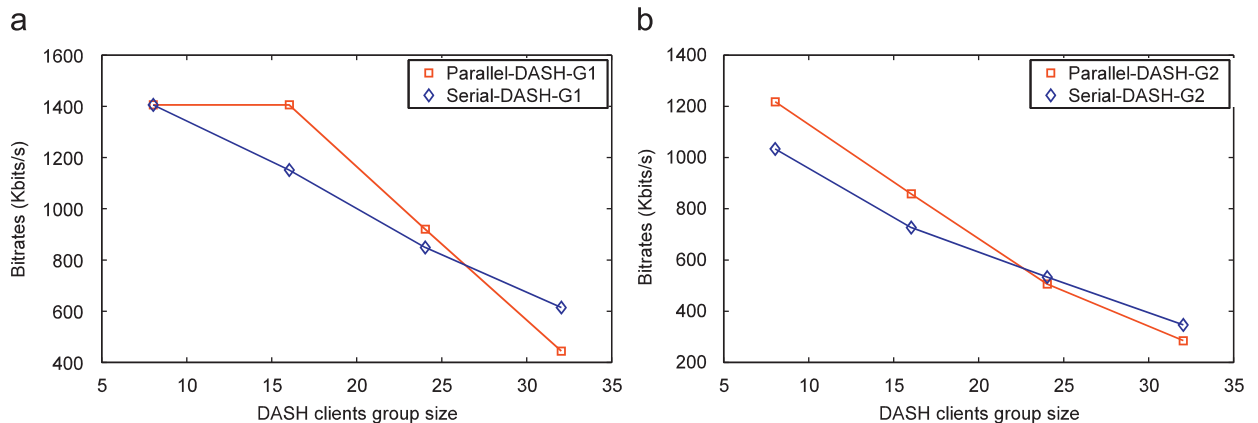


Fig. 14. Mean of average bitrates of two groups with edge server switching frequency 1: (a) the first group in 400–800 s and (b) the second group in 400–800 s.

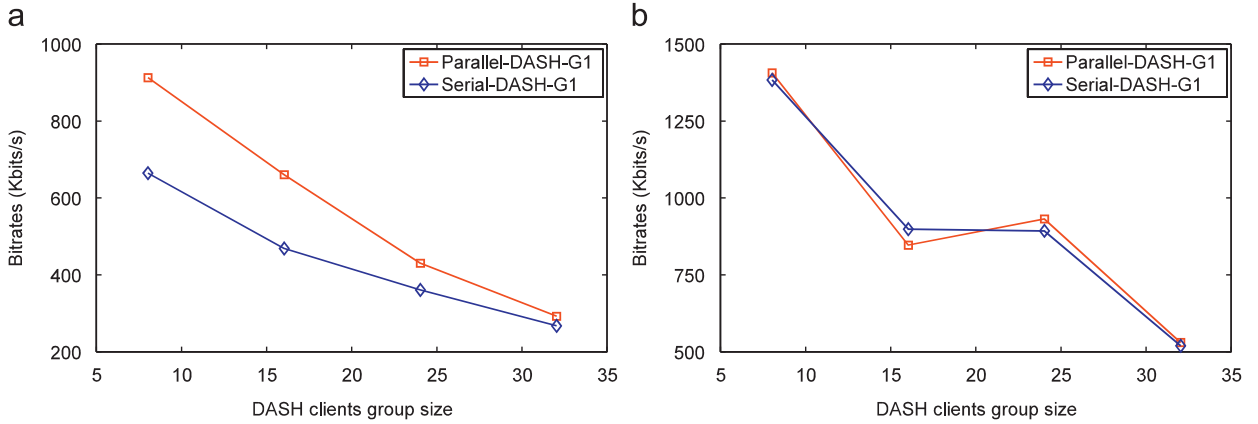


Fig. 15. Mean of average bitrates of the first group with edge server switching frequency 0.25: (a) the first group in 0–400 s and (b) the first group in 800–1200 s.

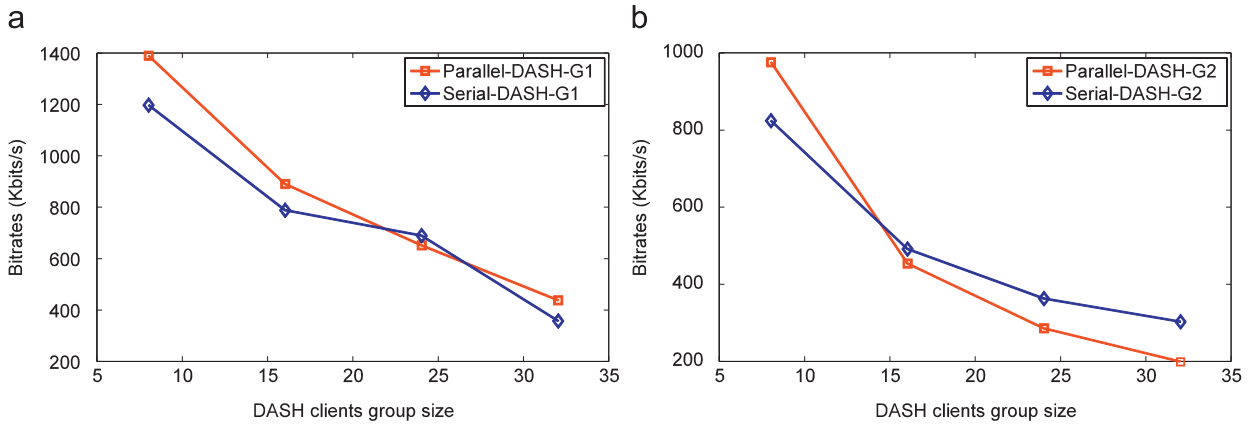


Fig. 16. Mean of average bitrates of two groups with edge server switching frequency 0.25: (a) the first group in 400–800 s and (b) the second group in 400–800 s.

method in the terms of convergence in the representation level. Comparing the rate adaptation performance with respect to the different edge server switching frequency, no significant difference is observed.

8.3. Average bitrates with varying group size

The mean of the average media bitrates for the different group sizes are reported with the bandwidth setting #1, for analyzing the rate adaptation algorithms with the respect to the achievable media bitrates under the conditions of the different crowded networks. In Figs. 13–16, the y and x axes denote the mean of the average media bitrates of each client group and the group size, respectively. And the Serial-DASH-G1/2 and the Parallel-DASH-G1/2 denote the rate adaptation results for the serial and parallel segment fetching methods in the first/second group, respectively. Fig. 13(a)–(b) shows the mean of the media bitrates of the first group in the period of 0–400 s and 800–1200 s, respectively. In both periods, the distributed networks bandwidths are only used by the clients of the first group without competition from the second group. So the number of active clients in the period of 0–400 s and

Table 4
Buffer underflow frequency.

Group size	Buffer underflow frequency			
	Edge server switching frequency 1		Edge server switching frequency 0.25	
	RA for the serial	RA for the parallel	RA for the serial	RA for the parallel
8	0	0	0	0
16	0	0	0	0
24	0	0	0	8
32	0	2	0	15

800–1200 s is half of the group size. Fig. 14(a)–(b) shows the mean of the media bitrates of the first and the second group in the period of 400–800 s. In this period, the distributed bandwidths are shared by the clients of the two groups, so the number of active clients is equal to the group size. Comparing the rate adaptation algorithms for the serial and the parallel segment fetching methods, the mean of the media bitrates of the rate adaptation algorithm for the parallel segment fetching method is higher than that of the

serial segment fetching method in most of the cases. Comparing the results in Figs. 13 and 14, the achievable means of the average media bitrates in the period of the 0–400 s and 800–1200 s are larger than those in the period of 400–800 s. The reason for this is that the clients of the second group become active in the period 400–800 s. In addition, the mean of the media bitrate decreases along with an increase in the group size in the rate adaptation algorithms for the serial and parallel segment fetching methods.

To evaluate the efficiency of the proposed rate adaptation methods with different edge server switching frequencies, simulation results with the edge server switching frequencies of 0.25 are reported in Figs. 15 and 16. Simulation results show that the performance of the achievable media bitrates decreases in the rate adaptation algorithms for the serial and the parallel segment fetching methods when the edge

server switching frequency is changed from 1 to 0.25. The main reason is that the frequent edge server switching strategy balances the loads of the distributed edge servers better than the infrequent edge server switching strategy. It indicates that the rate adaptation algorithms depend on the performance of the load balancing method, hence it is worthy to develop an advanced load balancing algorithm and edge server selection method for improving the performance of DASH especially with respect to achievable media bitrates.

8.4. Buffer underflow frequency

Table 4 reports the frequency of the buffer underflow for various group sizes. The edge server switching frequencies include 1 and 0.25. In Table 4, the rate adaptation algorithms for the serial and the parallel segment fetching

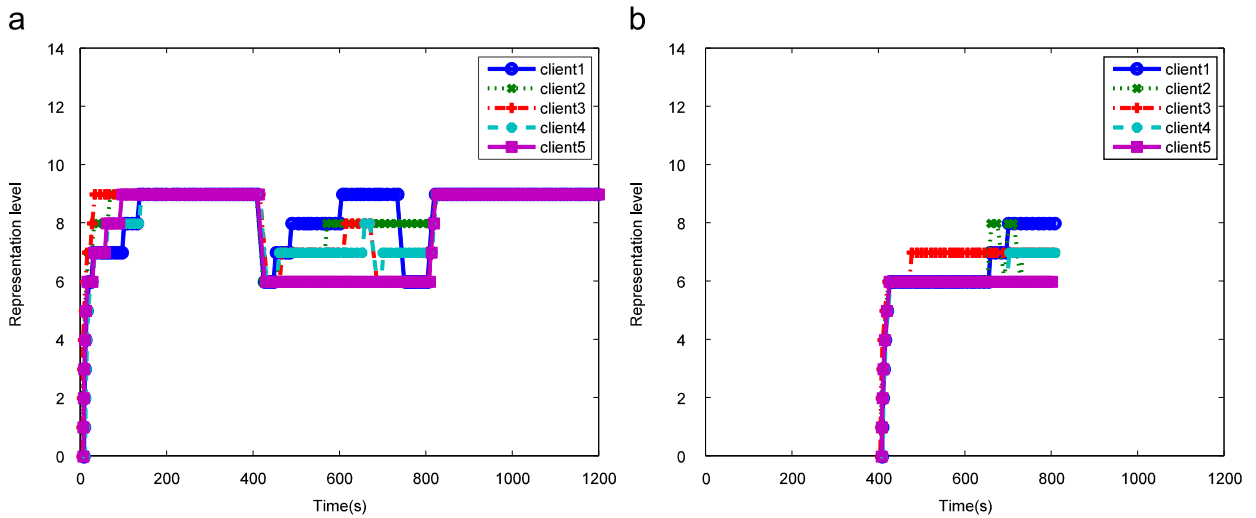


Fig. 17. Representation level evolution of the rate adaptation algorithm for the serial segment fetching method with group size 10 and bandwidth setting #2: (a) clients of the first group and (b) clients of the second group.

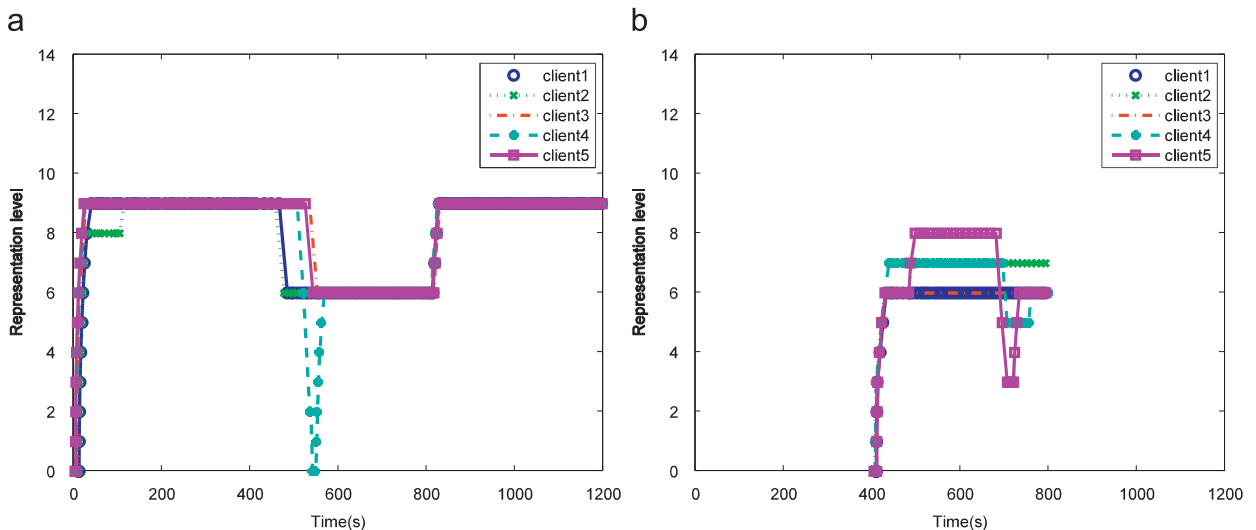


Fig. 18. Representation level evolution of the rate adaptation algorithm for the parallel segment fetching method with group size 10 and bandwidth setting #2: (a) clients of the first group and (b) clients of the second group.

methods are uniformly noted as RA. Buffer underflow has not occurred in the rate adaptation algorithm for the serial segment fetching method. However, buffer underflow occurred twice and 23 times in the rate adaptation algorithm for the parallel segment fetching method with edge server switching frequency 1 and 0.25, respectively. Simulation results indicate that the underflow frequency depends on the edge server switching frequency. In-frequent edge server switching cause a higher interruption frequency in the clients compared to frequent switching. The reason is that frequent edge server switching provides a more advanced load balancing algorithm compared to in-frequent switching. The target of the parallel segment fetching method is to improve the achievable media bitrates by fully utilizing the distributed edge server bandwidths. To improve the achievable media bitrate, however, an advanced load balancing algorithm is required. Without an advanced load balancing algorithm, the rate adaptation algorithm for the parallel segment fetching method may suffer from a relatively higher buffer underflow as shown in Table 4. The parallel segment fetching method improves the achievable media bitrates as reported in the simulation results in Section 8.3. So it is worthy to further exploit the load balancing algorithm of the CDN to improve the performance of DASH rate adaptation algorithm and further enhance the media streaming experience perceived by users.

8.5. Representation level evolution

In this sub-section, the representation level evolutions are reported to demonstrate the instantaneous changes of the representation level over time. The network topology #2 shown in Fig. 7(b) and the bandwidth setting #2 are used to report the representation level evolution. The edge server switching frequency is set to 0.25. There are five clients in the first and the second group. To evaluate the inter-client effect, the proposed rate adaptation algorithm behavior is compared with the rate adaptation behavior of Smooth streaming reported in [15].

Figs. 17 and 18 show the representation level evolution of the rate adaptation algorithms for the serial and the parallel segment fetching methods, respectively. The y and x axes denote the instantaneous representation level and the simulation time, respectively. In both figures, the results of the rate adaptation algorithm in the first group and the second group are reported in (a) and (b), respectively.

First, the rate adaptation speed can be observed from the representation level evolution. The rate adaptation speed is one of the most important factors for evaluating the efficiency of the rate adaptation algorithm for DASH. A fast switch-down prevents the client buffer underflow and a fast switch-up affects playback media quality. In the period of 0–400 s of Figs. 17-(a) and 18(a), the representation level can switch-up from level 0 to level 6 in less than 20 s and can switch-up to the highest level in less than 30 s, respectively. In the period of 400–800 s, the networks become congested because the clients of the second group start DASH randomly at the period of 400–410 s. In this period as shown in Fig. 17(a), the clients of the first group and the second group switch-down and switch-up to the appropriate representation level within

30 s. In the period of 400–800 s as shown in Fig. 18(a), some clients have switched-down to a lower layer by taking more time compared to the rate adaptation algorithm of the serial segment fetching method in Fig. 17(a). However, clients of the second group in Fig. 18(b) are able to switch-up to a higher level as fast as in Fig. 17(b). In the period of 800–1200 s of Figs. 17 (a) and 18(a), after removing the clients of the second group at 800 s, clients of the first group will reach to the highest level using less than 20 s. The representation level evolution results show that the proposed rate adaptation algorithms quickly switch-up and switch-down representation levels by efficiently identifying spare network capacities and network congestion. If spare network bandwidths are available, fetching segment takes less time compared to the optimum segment fetch time, i.e. *ESFT*. Otherwise, if the networks become congested, segment fetch time is larger than *ESFT*. Hence, the proposed rate adaptation method can switch-up and switch-down representation level quickly.

Second, the convergence behavior, fairness between different clients and inter-client effect can be evaluated through the evolution of the representation levels. In the period of 0–400 s and 800–1200 s of Figs. 17(a) and 18(a), the proposed rate adaptation algorithms for the serial segment fetching method and the parallel segment fetching method show an improved convergence and a better fairness between clients in the first group. In the period of 400–800 s, Figs. 17(a) and 18(a) show that clients of the first group will switch-down to a lower level to react to the newly joining clients of the second group. And Figs. 17(b) and 18(b) show that clients of the second group will switch-up to the appropriate higher representation level.

In Fig. 17(a) and (b), clients of the first and the second group converge to the levels 6–9, which indicates that the proposed rate adaptation algorithm provides a decent convergence and fairness between different clients. Close to time 600 s, a switch-up occurred for several clients in the first group, which resulted in other clients having to switch-down to a lower level. For example, a switch-down occurred in the client#3 and client#4 in the first group. When multiple clients compete for the sharable bandwidth, improving convergence becomes a challenging task as discussed in Section 8.2. As shown in the simulation results, a severe hopping of the representation levels is avoided using the proposed *SFTM* and the conservative switch-up and switch-down factor given in Eqs. (14) and (18), respectively.

In Fig. 18(a) and (b), most of the representation levels of the first and the second group converge to levels 6–9. In Fig. 18(a), a client switches-down to a lower level near 550 s but it quickly switches-up to a higher level. Once congestion is detected, the proposed rate adaptation method deploys the multi-step based switch-down strategy, which may cause over switching-down. To prevent playback interruption in DASH due to client buffer underflow, the rate adaptation for DASH should be equipped with a faster switch-down speed compared to the rate adaptation for the RTP/UDP based streaming scenario. In media streaming over HTTP, congestion will cause a TCP congestion window, a TCP throughput decrease, and packets retransmission. It will further result in serious draining of the buffered media

time in the client's buffer. In case of over switching-down, the rate adaptation method can switch-up the representation level to the optimum level with a fast speed by using the proposed rate adaptation metrics as shown in the results in client #4 near 550 s. To further improve convergence, the sliding window strategy is used in this paper to measure the multiple rate adaptation metrics for switching-up. Each rate adaptation metric represents the reception of a segment or a portion of a segment, i.e. portion, which can be fetched in parallel from the distributed edge servers. The observation of the multiple segments or portions reception can efficiently represent the current bandwidths capacities in the distributed networks. Meanwhile, in order to avoid affecting the speed of switching-up, the representation levels within the measuring window can be different. So consecutive switching-up can occur as shown at the starting phase of DASH in all clients and near 550 s for client #4.

As mentioned in the previous Section 8.2, when two Smooth streaming players compete for the common link bandwidth, the second player, which starts later than the first player, fails to switch-up to the appropriate higher level according to the experimental results and evaluation reported in section 6 of [15]. Instead, the second Smooth streaming player oscillates between the lower levels of bitrates, which indicates the unfairness between the first and the second player. The failure of switching-up in the later joined client in the Smooth streaming may be incurred as the rate adaptation algorithm deploys the buffered media time to decide switch-up and switch-down. Hence, if a player has not buffered enough media time, it will fail to switch-up to the appropriate high level, which causes the unfairness.

In contrast, the reported samples of representation level evolutions in Figs. 17(b) and 18(b) show that clients of the second group, which start later than clients of the first group, can successfully switch-up to the higher representation level. Simulation results of Figs. 17(b) and 18(b) demonstrate that the proposed rate adaptation algorithms can efficiently reduce unfairness. Ten clients compete for the bandwidth in our simulation, while, two clients compete for the bandwidth in [15]. Thus the competition between different clients in this paper is more severe. The severe competition makes it more difficult for the clients to switch-up and converge because of the severe inter-client effects. Although the simulation conditions used are somewhat restricted, the proposed rate adaptation algorithms show a fast switch-up speed and an acceptable convergence behavior.

Last, buffer underflow does not occur during the whole period for clients of the first and the second group. In this paper, *RSFT*, which reveals the buffered media time status, is taken into account to determine *ESFT* to prevent the buffer underflow in a severe congestion.

9. Conclusion

In this paper, we propose two rate adaptation metrics and two rate adaptation algorithms for Dynamic Adaptive Streaming over HTTP (DASH). In DASH, segments can be requested one after another, called serial segment fetching method, or they can be requested in parallel, called parallel segment fetching method. A novel rate adaptation

metric is proposed, which can detect spare networks capacity and network congestion. The proposed rate adaptation metric compares the expected segment fetch time (*ESFT*) with the measured segment fetch time and determines if the current media bitrate matches the network capacity. The optimum segment fetch time is decided by the ideal segment fetch time (*ISFT*) and the real-time remaining segment fetch time (*RSFT*). The *ISFT* is determined by the media segment duration (*MSD*) multiplied by the parallel HTTP threads to deliver segments. The *RSFT* depends on the reception history of previous segments and is related to the buffered media time in the client buffer. Based on the proposed rate adaptation metric, two novel rate adaptation algorithms for the serial and the parallel segment fetching methods are presented in this paper. A sliding window strategy is used to measure multiple rate adaptation metrics to reduce the fluctuation frequency in the representation levels due to the different bandwidths in the different edge servers distributed in the networks, while not affecting switch-up speed. Furthermore, this paper investigates the fairness issue between different DASH clients which compete for the common link bandwidth. A method of solving unfairness is proposed which uses the priority based *RSFT* calculation method in the beginning phase of DASH for the newly joining clients. Extensive simulation results are reported by simulating DASH in CDN with multiple competing DASH clients. Simulation results show that the efficiency of the proposed rate adaptation algorithms in the aspects of switch-down and switch-up speed, convergence behavior and fairness between different DASH clients. The rate adaptation algorithm for the parallel segment fetching method outperforms the rate adaptation algorithm for the serial segment fetching method with respect to achievable media bitrates but slightly inferior with respect to convergence and buffer underflow frequency. In addition, the achievable media bitrates of the rate adaptation algorithm for the serial and the parallel segment fetching method depends on the load balancing algorithm. For the parallel segment fetching method, the buffer underflow frequency of the rate adaptation algorithm relies on the on the load balancing algorithm. In future work, we will investigate further CDN optimization for DASH.

Acknowledgment

This work was supported by the Academy of Finland, (application number 129,657, Finnish Programme for Centres of Excellence in Research 2006–2011). The authors would like to appreciate the efforts of the anonymous reviewers for their comments which have improved the paper.

References

- [1] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol-HTTP/1.1, RFC 2616, June 1999.
- [2] C. Krasic, K. Li, J. Walpole, The case for streaming multimedia with TCP, in: Proceedings of the IDMS, Lancaster, UK, September 2001.

- [3] B. Wang, J. Kurose, P. Shenoy, D. Towsley, Multimedia streaming via TCP: an analytic performance study, *ACM Transactions on Multimedia Computing Communications and Applications (TOMCCAP)* 4 (2) (2008).
- [4] T. Kim, M.H. Ammar, Receiver buffer requirement for video streaming over TCP, in: *Proceedings of the SPIE VCIP*, 2006, San Jose, CA, January 2006.
- [5] A. Zambelli, IIS Smooth Streaming Technical Overview, Microsoft Corporation, 2009.
- [6] Adobe, HTTP Dynamic Streaming on the Adobe Flash Platform, Adobe Systems Incorporated, 2010.
- [7] 3GPP TS 26.234 Release 9: Transparent end-to-end Packet-switched Streaming Service (PSS), protocols and codecs.
- [8] ISO/IEC 23009-1: Dynamic Adaptive Streaming Over HTTP (DASH)-Part 1: Media Presentation Description and Segment Formats, Draft International Standard, August 30 2011.
- [9] 3GPP TS 26.247 Release 10: Transparent end-to-end Packet-switched Streaming Service (PSS), Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH).
- [10] C. Liu, I. Bouazizi, M. Gabbouj, Rate adaptation for adaptive HTTP streaming, in: *Proceedings of the ACM Multimedia Systems Conference (ACM MMSys 2011)*, San Jose, California, February 23–25, 2011.
- [11] C. Liu, I. Bouazizi, M. Gabbouj, Parallel adaptive HTTP media streaming, in: *Proceedings of the IEEE International Conference on Computer Communications and Networks (ICCCN 2011)*, Hawaii, July 31–August 4, 2011.
- [12] S. Lam, J.Y.B. Lee, S.C. Liew, W. Wang, A transparent rate adaptation algorithm for streaming video over the internet, in: *Proceedings of the Eighteenth International Conference on Advanced Information Networking and Applications*, Fukuoka, Japan, March 2004.
- [13] N. Färber, S. Döhla, J. Issing, Adaptive progressive download based on the MPEG-4 file format, *Journal of Zhejiang University Science A* 7 (Suppl. 1) (2006) 106–111 (*Proceedings of International Packet Video Workshop 2006*).
- [14] R. Kuschnig, I. Kofler, H. Hellwagner, An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC, in: *Proceedings of the ACM Multimedia Systems Conference (ACM MMSys 2010)*, Scottsdale, Arizona, February 2010.
- [15] S. Akhshabi, A. Begen, C. Dovrolis, An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP, in: *Proceedings of the ACM Multimedia Systems Conference (ACM MMSys 2011)*, San Jose, California, February 23–25, 2011.
- [16] R. Kuschnig, I. Kofler, H. Hellwagner, Evaluation of HTTP-based request-response streams for internet video streaming, in: *Proceedings of the ACM Multimedia Systems Conference (ACM MMSys 2011)*, San Jose, California, February 23–25, 2011.
- [17] R. Kuschnig, I. Kofler, H. Hellwagner, Improving internet video streaming performance by parallel TCP-based request-response streams, in: *Proceedings of the Seventh Annual IEEE Consumer Communications and Networking Conference (IEEE CCNC 2010)*, January 2010.
- [18] S. Tullimas, T. Nguyen, R. Edgecomb, S. Cheung, Multimedia streaming using multiple TCP connections, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)* 4 (2) (2008).
- [19] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. Hansen, P. Engelstad, Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks, in: *Proceedings of the ACM Multimedia Systems Conference (ACM MMSys 2011)*, San Jose, California, February 23–25, 2011.
- [20] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally distributed content delivery, *IEEE Internet Computing* (2002) 50–58.
- [21] B. Krishnamurthy, C.E. Wills, Y. Zhang, On the use and performance of content distribution networks, in: *Proceedings of Internet Measurement Workshop*, 2001.
- [22] V.K. Adhikari, S. Jain, Z. Zhang, Where do you "tube"? uncovering YouTube server selection strategy, in: *Proceedings of the IEEE International Conference on Computer Communications and Networks (ICCCN 2011)*, Hawaii, July 31–August 4, 2011.
- [23] G. Pallis, A. Vakali, Insight and perspectives for content delivery networks, *Communications of the ACM (CACM)* 49 (No. 1) (2006) 101–106.
- [24] C. Liu, I. Bouazizi, M. Gabbouj, Segment duration for rate adaptation of adaptive HTTP streaming, in: *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2011)*, Barcelona, Spain, July 11–15, 2011.
- [25] The network simulator Ns-2. Available [online]. <<http://www.isi.edu/nsnam/ns>>.
- [26] C. Müller, C. Timmerer, A test-bed for the dynamic adaptive streaming over HTTP featuring session mobility, in: *Proceedings of the ACM Multimedia Systems Conference (ACM MMSys 2011)*, San Jose, California, February 23–25, 2011.