A Pure Network-Based Approach to Achieve Always Best Quality Video Streaming

Toan Nguyen-Duc(✉), Xuan Doan-Thanh, and Hung Do-Viet

Hanoi University of Science and Technology, Hanoi, Vietnam
xuan.doanthanh@hust.edu.vn

Abstract. The demand for video streaming has been more and more increasing, causing the streaming technology and the related technologies to be improved to meet the requirement of the best quality of experience (QoE) from various users. A lot of research has been focusing on studying users' behavior or developing streaming client and/or server application. These works estimate the network state passively and are lack of a global view of the network. As a result, they meet difficulty in bandwidth competition, QoE fairness scenarios. Some works optimize routing mechanism to improve video quality and QoE. This work also proposes a pure network-based approach, however taking into account the characteristics of video streaming application, to support an always best QoE to end-users. The proposed approach leverages the advantage of SDN network to convert the explored characteristic of streaming application into the network configuration. The proposal has been implemented on a real testbed. The obtained results show that the proposed mechanism has maintained the best video quality while maximizing the bandwidth for competitor application, i.e., file download using ftp protocol.

Keywords: HTTP adaptive streaming · QoE fairness · Bandwidth competition · Software-defined networks

1 Introduction

Video streaming is one of the most popular services on the Internet. The demand for high-quality videos is growing exponentially. Cisco [1] estimates that the amount of video traffic will be about 37 exabytes (EB) per month. Therefore, video streaming techniques have been examined and improved. Along with the demands for high-quality videos is the demand for the best Quality of Experience (QoE). This is still an open challenge for the Internet network manager since the Internet was originally designed for the best-effort, non-real-time data transmission.

To this end, many research efforts focus on studying users' behavior, improving system architecture, proposing handover mechanisms, developing video streaming client and server application, optimizing network routing algorithms. It is clear that to understand the user experience directly is the most straight and effective way to improve QoE. However, this method may violate the users' privacy. Several researches [1–3] have

Y. Weng et al. (Eds.): TridentCom 2020, LNICST 380, pp. 47–58, 2021.
https://doi.org/10.1007/978-3-030-77428-8_4

studied the behavior of adaptive streaming application to provide an adaptive bitrate mechanism. The authors in [3] rely on a special mobility server acts as a video cache server at the edge network to adjust the video bitrate. The cached video may not contain all quality versions, leading video clients to download popular downloaded video segments, not their desired ones. The authors in [1, 2] estimate the network state based on the buffer on the client side. These works estimate the network state passively and lack of a global view of the network. As a result, they meet difficulty in bandwidth competition, QoE fairness scenarios.

It is possible to control quality of video streaming application only from network domain. The authors in [4] proposed a software-define infrastructure to assist the adaptive bitrate in video streaming. Their proposal monitors the state of network device to select the best cache in the network. Then, they modify the adaptive bitrate algorithm on the video client. This work did not consider the scenario where the video streaming application has to compete bandwidth with other application. For this purpose, many researches [5–7] have tried to maximize the QoE of multiple clients in a shared network environment. The authors in [5] focused on QoE fairness among multiple video clients. The video clients run on devices with various resolutions, hence the received experience for users are quite different. The work in [6] considered the scenario where a large number of video client share the same network. An external application was developed to communicate with the controller about the requirement per video client. The work in [7] is the most relevant to our proposal. The authors [7] consider the bandwidth competition between HAS and other application. They focus on estimating the QoE based on the collected QoS values. The estimated QoE is then used to control the network. However, this work did not consider the characteristic of the HAS application in controlling the network.

Different from existing studies, this work explores the details of HAS application behavior to provide a suitable network control, aiming to maintain the best QoE for video player while maximizing the available bandwidth for other application. The proposed mechanism has been implemented on a real testbed. The experiments represent a reality situation where video traffic has to compete for bandwidth with greedy TCP flows, i.e., file download. The obtained results show that the proposed mechanism satisfy both type of users. The video was always delivered with the best quality. Also, the time for downloading file was 15.5 s. This duration is shorter more than ten percent in comparison with the case our mechanism was not used.

The rest of the paper is organized as follows: In Sect. 2, the background of this work including basics of HTTP Adaptive streaming, the behavior of HAS application and SDN technologies will be described. In Sect. 3, the proposed algorithm will be explained. Section 4 will describe the evaluation on the performance of the system. Finally, the paper will be concluded in Sect. 5.

2 Background

2.1 HTTP Adaptive Streaming (HAS)

In HAS, video contents are split into small segments of a certain length. Each segment is encoded in several predefined qualities. A combination of resolution, frame rate,

bitrate, etc., determines such quality. MPEG-DASH describes these characteristics of the segments in a file called Media Presentation Description (MPD), which is an Extensible Markup Language (XML) file. The segments and the MPD file are then hosted on the server. Before a streaming session is started, the client requests the MPD file through an HTTP GET request. Based on the information in this file, segment requests can be sent to the media server. To request a suitable segment size, the client has to measure the available bandwidth to estimate the condition of the end-to-end connectivity. Based on the estimated available bandwidth and playout buffer conditions, the client determines the suitable bitrate and requests the appropriate video segments by sending HTTP GET requests. Once a segment is entirely downloaded, the next segment is requested. The received segments are individually decoded and are played in sequence.

2.2 Exploring HAS Application's Behavior

When an end-user is watching HAS-based videos, video segments are regularly downloaded. To download a segment, the client sends a GET request to the server. If the client detects that the network throughput is getting worse enough, i.e., below a threshold, then the resolution and bitrate of the video segment in the GET request will reduce.

In scenario A, a measurement of video streaming in a stable state is conducted. Here, the stable state is defined that there is no competition from other users, only one network user, and the network capacity is fast enough to stream videos at the highest speed. The considered metric is the average duration between 2 consecutive GET requests. The obtained average duration between 2 consecutive GET requests is 4.003 s.

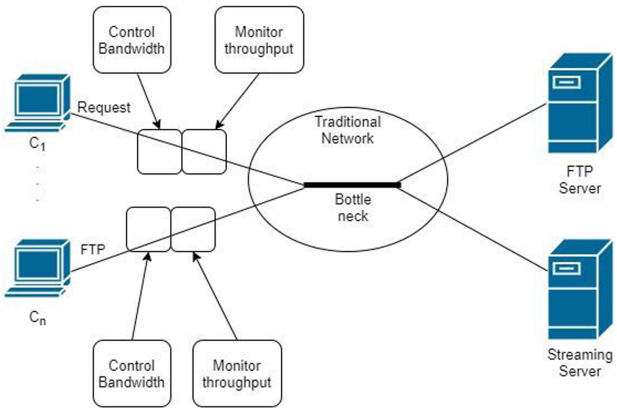


Fig. 1. HAS competes bandwidth in a traditional network

In scenario B, video streaming and other TCP-based application, i.e., File download using FTP protocol, generally share a bottleneck link; hence, they have to compete for bandwidth as shown in Fig. 1. The measured results are given in Fig. 2, the video quality was reduced from 1920 × 1080 pixels to 1280 × 720 pixels. Also, the average duration between 2 consecutive GET requests changed to 7.17 s. The average duration

to download a 10-MB file was 17.29 s. This download duration is 9 s longer than that obtained when the file was downloaded in the stable state.

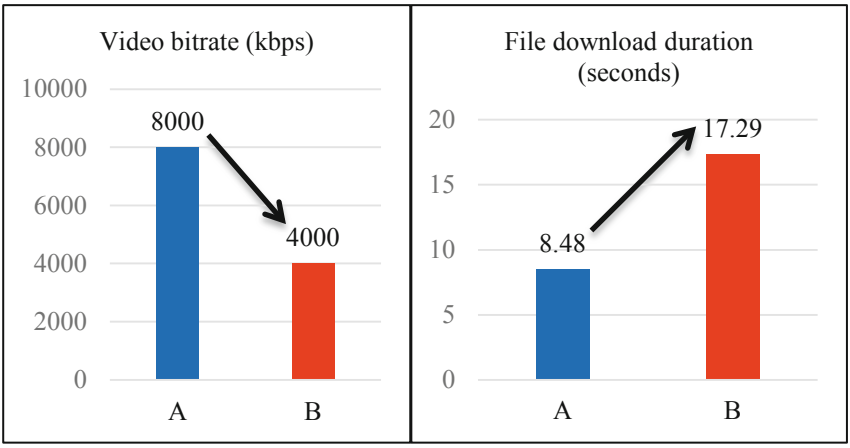


Fig. 2. HAS behavior in two scenarios: A. Stable state and B. Bandwidth competition

2.3 Software-Defined Network (SDN)

In an SDN-based network, the SDN controller knows the network wide information including the network topology as given in Fig. 3. Hence, it potentially allocates a suitable bandwidth for each connection. The problem is that the SDN controller only works with SDN switch, it does not communicate with the hosts. Therefore, the SDN controller is unable to understand the hosts' requirement. This work will propose a mechanism to address this issue.

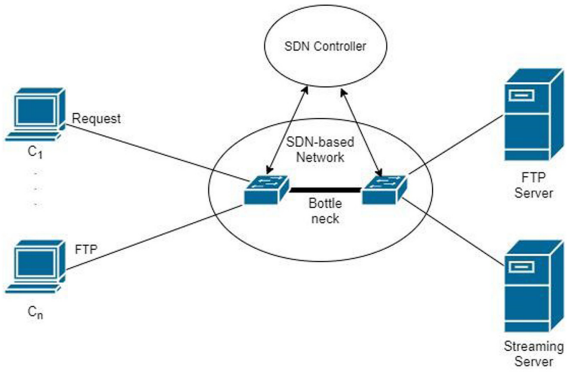


Fig. 3. HAS in an SDN-based network

3 Our Proposed Mechanism

As mentioned in Sect. 2.1, during a streaming process, the video client periodically sends GET requests to the server. Each GET request is sent in a packet. In each packet, parameters including video bitrate and video resolution are requested. The value of the required parameters is based on the estimation of the connectivity state, i.e., the estimated available bandwidth.

Once the GET request is sent, if the requested video segment is available, it will be downloaded to the client. In general, all video segments have the same duration, however, each segment has a different frame rate causing the size of each segment to be different. Therefore, the duration to download a segment will vary depending on the size of the segment. Short segments are good to adapt quickly to bandwidth changes and prevent stalls, but longer segments may have a better encoding efficiency and quality. The segment length recommended for DASH is around 2 to 4 s [8]. Therefore, a duration of 4 s has been chosen as the segment length, which is a good compromise between encoding efficiency and flexibility for stream adaption to bandwidth changes.

The HAS application behavior will be modeled in the following section.

3.1 Modeling HAS Application Behavior

To capture application traffic, every packet has been sniffed. All the packets belonging to the monitored application are filtered. The collected information is stored in text files. The monitoring procedure for the GET requests is as follows (Fig. 4):

```
GET interface from keyboard
IF interface:
  sniff(filter="port 80", prn=process_packet, iface=iface)
ELSE using default interface:
  sniff(filter="port 80", prn=process_packet, store=False)
ENDIF

FUNCTION process packet(packet):
IF packet is an HTTP Request:
  Get the requested URL
  Get the source and destination IP Address of the requester
  Get the request method
  method <- packet[HTTPRequest].Method.decode
  dateTimeObj <- now
  Turn up a timer
  IF method is "GET":
      Write the packet's information to a file
  ENDIF
  ENDFUNCTION
```

Fig. 4. Pseudo code of the monitoring procedure

After the duration between 2 consecutive GET requests has been studied. The video streaming behavior within a duration is modeled as illustrated in Fig. 5.

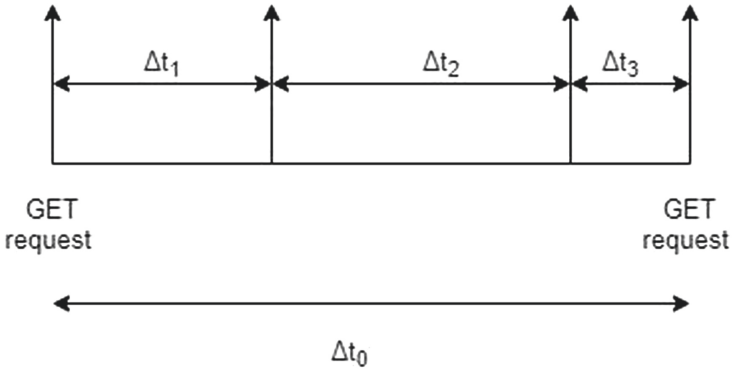


Fig. 5. Video streaming behavior

The Fig. 5 shows that Δt_0 is the duration between 2 consecutive GET requests. Within Δt_0, Δt_1 is the duration for the client to download the segment, Δt_3 is the duration for the client to detect connectivity condition, and Δt_2 is the duration that the video client may not consume the network bandwidth.

3.2 Considered Bandwidth Competition Scenario

In this work, a client requests the video resolution from a video streaming server. It is assumed that the bandwidth competition occurs after the client sends request for the highest video quality as shown in Fig. 6.

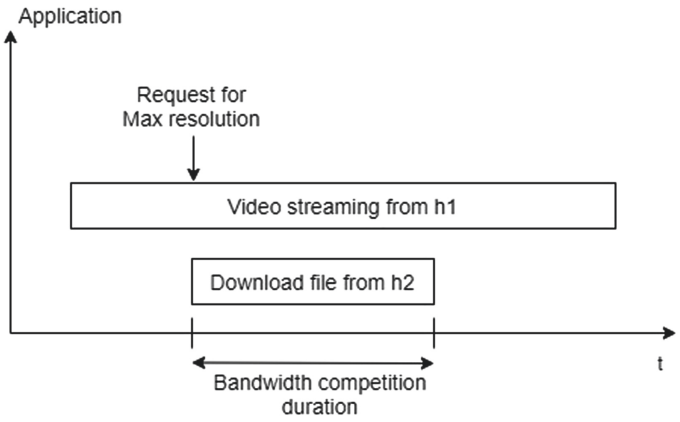


Fig. 6. Considered scenario

The duration of file downloading is assumed to be not as long as the same mechanism could be applied repeatedly for long duration cases. Due to the native TCP congestion control, all applications will have to share the available bandwidth fairly. As a result, the

video client may have to request a lower video resolution if the bandwidth is lower than a threshold. Also, the duration to download file is longer.

3.3 Proposed Mechanism in Bandwidth Competition Scenario

The proposed algorithm to control network in case of bandwidth competition is given in Fig. 7. While video is streaming from the video server to the video client, a module written in python is to monitor video segment requests from the video client. When the maximum video resolution has been requested, the module to handle bandwidth competition is invoked. The module bandwidth competition handle takes the model of application behavior as the input to translate into network configuration to control the network in 3 consecutive GET requests.

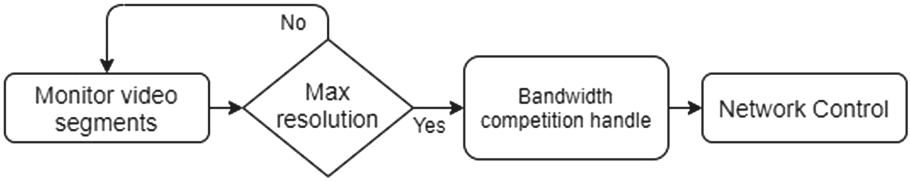


Fig. 7. Algorithm to control network in bandwidth competition scenario

The mechanism of the bandwidth competition handle is as follows: After the video client sends the GET request for the maximum resolution, it is necessary to provide enough bandwidth for the video client to download the segment. In this duration, the bandwidth for the other application will be limited. Once the video segment is downloaded, the bandwidth for the video application can be reduced. This duration is an opportunity for other applications to have maximized available bandwidth. The bandwidth allocation for the video application must be increased before the client starts detecting network conditions. This means bandwidth for other applications must be reduced.

4 Evaluation

4.1 Experiment Setup

The topology of the experiments is shown in Fig. 8.

In this topology, three hosts emulated by Mininet connect to a real server via an emulated switch s1. The switch can be controlled by an SDN controller, however, in this work, the network operates without any controllers. The interface eth0 of the host h1, the interface eth0 of the host h2, and the interface eth0 of the host h3 are connected to port 1, port 2, port 3 of the switch s1, respectively. The bandwidth between h1 and s1 is B1−1, the bandwidth between h2 and s1 is B1−2, and the bandwidth between h3 and s1 is B1−3. The bandwidth between s1 and the server is Bs-s. Each of this bandwidth can be set individually to any values. Initially, all of them are set to 10 Mbps.

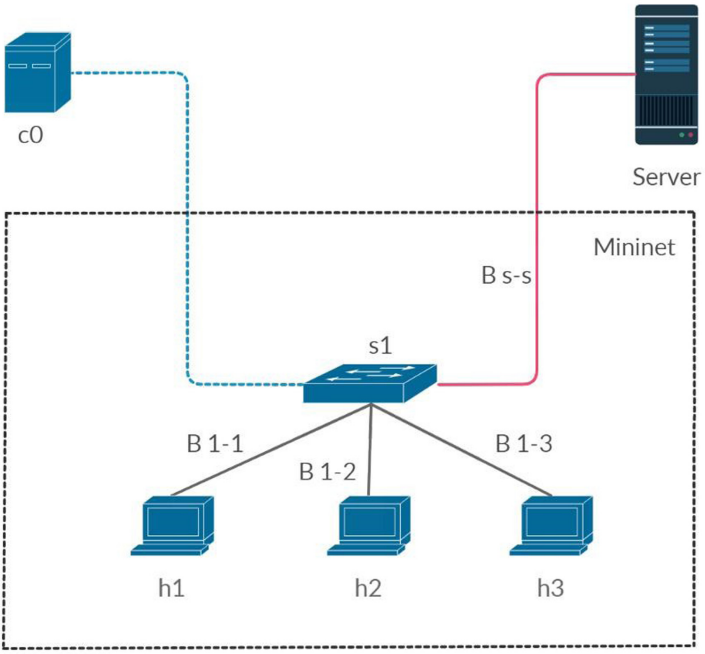


Fig. 8. Experiment topology

Figure 8 also shows that hosts emulated by Mininet connect to a real server via an emulated switch. Mininet hosts can communicate with a real server by using NAT technology. The operating system of the server is Windows version 10 64bit. On the server, the built-in Internet Information Services (IIS) is used as a Web Server to stream video via HTTP protocol. The videos used in the experiment is from [9] have characteristics as given in Table 1.

To stream video, two tools, namely MP4Box and MP4Client, of an open-source framework called GPAC [10], have been used. MP4Box was used for the preparation of HTTP Adaptive Streaming content; while MP4Client is used as a video player. The model to stream video adaptively from a server to a client is described in Fig. 9. The video contents generated by the video streaming server are put into a web server. These contents are streamed to the video player at the client via HTTP protocol.

Table 1. Video file characteristics

Filename	Frame rate	Resolution	Bit rate
bbb_30fps_320 × 180_200k.mp4	30 fps	320 × 180 pixels	200 kbps
bbb_30fps_320 × 180_400k.mp4	30 fps	320 × 180 pixels	400 kbps
bbb_30fps_480 × 270_600k.mp4	30 fps	480 × 270 pixels	600 kbps
bbb_30fps_640 × 360_800k.mp4	30 fps	640 × 360 pixels	800 kbps
bbb_30fps_640 × 360_1000k.mp4	30 fps	640 × 360 pixels	1000 kbps
bbb_30fps_768 × 432_1500k.mp4	30 fps	768 × 432 pixels	1500 kbps
bbb_30fps_1024 × 576_2500k.mp4	30 fps	1024 × 576 pixels	2500 kbps
bbb_30fps_1280 × 720_4000k.mp4	30 fps	1280 × 720 pixels	4000 kbps
bbb_30fps_1920 × 1080_8000k.mp4	30 fps	1920 × 1080 pixels	8000 kbps

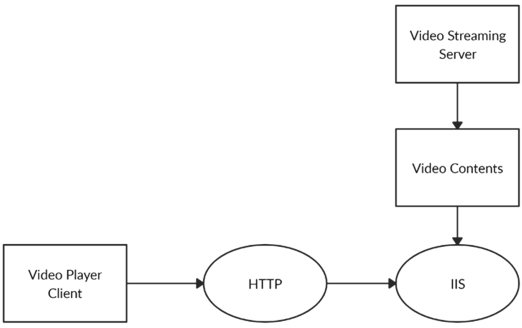


Fig. 9. Video streaming client-server model

The Video player clients in Fig. 9 run on hosts emulated by Mininet as shown in Fig. 8. The Mininet built on a computer installed Ubuntu Operating System version 18.04 64bit is used to emulate the simple and customized topology. To customize the bandwidth for each host-to-switch connection the TCLink library was used. There are 2 ways to control link bandwidth for a host in Mininet. The first one is to use Linux Traffic Control (qdisc) and the second method is to use TCLink library. Qdisc operates out of the Mininet, and it requires to shut down the link to assign new capability, then the link is turned on. In the latter, the control of the link bandwidth is easier because it is built-in in the Mininet.

4.2 Evaluate the Proposed Mechanism

The conducted experiment used the topology as illustrated in Fig. 8. The streaming was performed by the host h1, and file downloading was performed by the host h2. The bandwidth between the host h1 and the switch s1; the host h2 and the switch s1; the switch s1 and the server were all set to 10 Mbps.

To automate the experiment, a python program was developed to load Mininet environment with custom topology. The pseudocode of the python file is as follows:

```
Create a single switch topo
Create Mininet network
Add NAT technique to the network
Start the network
Get instance of hosts and switch
Set bandwidth to each link between host and switch
Start Command-Line interface of the network
```

Fig. 10. Pseudo code of creating emulated environment for the experiment

The procedure for this experiment is as follows:

Firstly, at the client-side, the python script described in Fig. 10 is used to start Mininet for network creation. At the same time, the bwm-ng tool is started to observe the receiving throughput over the network interface h1−eh0 of the host h1 and the network interface h2−eh0 of the host h2. Due to the simulation environment, we actually observe the network interface s1−eth1 and s1−eth2 of switch s1, which is directly connected to h1−eth0, h2−eth0, respectively.

On the server side, the video content has been placed in the web-server, waiting for request from video client. When a streaming session starts, its information will be captured by executing a shell script on the xterm terminal of the host h1.

When the monitoring module detects the first GET request with the maximum resolution of 1920 × 1080 pixels, which is the highest resolution available at the server, the file download is triggered. The file download information is also saved in a log file by executing a shell script on the xterm terminal of the host h2.

When the video segment is being downloaded, the bandwidth of file download is reduced so as not to affect the video stream. This was achieved by executing the following command:

```
py h2.connectionsTo(s1)[0][0].config(bw=1) , h2.connec-
tionsTo(s1)[0][1].config(bw=1), s1.cmd('sleep 4')
```

As shown in the command, the parameter bw is equal to 1 meaning that the bandwidth between h2 and s1 is going to be set to 1 Mbps. This bandwidth limitation only works for 4 s with the “sleep” command. During this time, the bandwidth of video streaming was 9 Mbps, which was sufficient for video quality not to be affected.

Once the video segment has been downloaded, the bandwidth for downloading the file is increased, the bandwidth for video streaming is reduced. This was achieved by executing the following command:

```
py h1.connectionsTo(s1)[0][0].config(bw=1) ,
h1.connectionsTo(s1)[0][1].config(bw=1) ,
h2.connectionsTo(s1)[0][0].config(bw=9) ,
h2.connectionsTo(s1)[0][1].config(bw=9), s1.cmd('sleep 9')
```

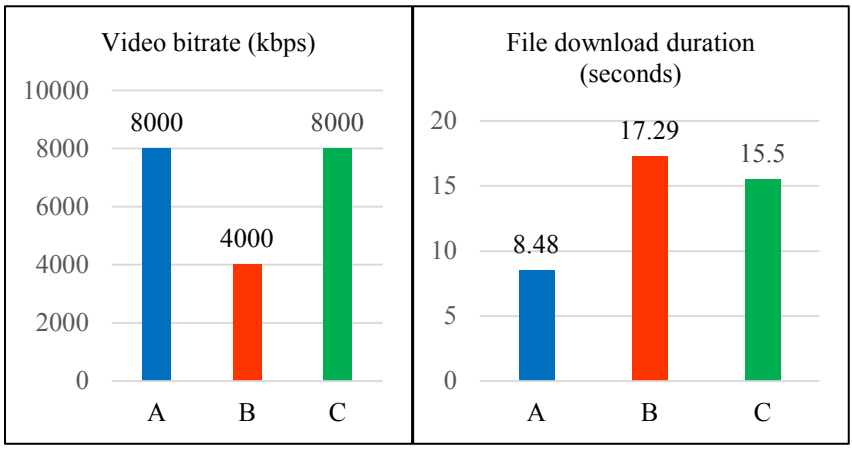
As shown in the command, bw = 1 means that the bandwidth between h1 and s1 was set to 1 Mbps; bw = 9 means that the bandwidth between h2 and s1 was set to 9 Mbps. This bandwidth limitation only works for 9 s with the "sleep" command. This is to make the video player not able to detect the bandwidth change.

Afterward, to prepare for the video client to make decision for the next GET request, the bandwidth for the video must be large enough so that there is no GET request with a lower resolution. For this purpose, the video bandwidth is increased, the bandwidth of the file download is reduced. This was achieved by executing the following command:

```
py h1.connectionsTo(s1)[0][0].config(bw=9) ,
h1.connectionsTo(s1)[0][1].config(bw=9) ,
h2.connectionsTo(s1)[0][0].config(bw=1) ,
h2.connectionsTo(s1)[0][1].config(bw=1)
```

This command adjusted the bandwidth between h1 and s1, the bandwidth between h2 and s1 to 9 Mbps and 1 Mbps, respectively. This returned enough bandwidth for h1 to have video with the highest quality.

The procedure was repeated 20 times. The obtained results show that the video quality was not reduced and the average file download duration was 15.5 s. Figures 11a and 11b compare the obtained results in three scenarios: A. Stable state, B. Bandwidth competition without our proposal, and C. Figure 11a shows that our proposed mechanism helps to maintain always the best video quality to the client even in case of bandwidth competition. Figure 11b shows that our proposed mechanism also helps the competitor application to have a faster connectivity than that when our mechanism is not applied. Specifically, the duration to download file was reduced more than ten percent from 17.29 s to 15.5 s.



a) b)

Fig. 11. Performance comparison

5 Conclusion

The goal of this work is to find a mechanism that guarantees the video quality and reduce the impact for the download file in case of bandwidth competition, has been achieved. The proposed mechanism has been implemented on a real testbed. The obtained results show that there is an 1.79-s improvement in reducing file download duration. This is about 10.35% as compared to the duration to download file in the bandwidth competition case. The important point is the quality of video streaming in the same experiment has been maintained.

In the future, this work will be extended to diverse scenarios where more clients participate in the network, more services compete for the bandwidth. Different types of video players and video streaming mechanisms will be explored. The control of the network will be executed from SDN controllers.

References

1. Akhshabi, S., Begen, A.C., Dovrolis, C.: An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In: Proceedings of the Second Annual ACM Conference on Multimedia Systems, pp. 157–168, February 2011
2. Nam, H., Kim, K.H., Calin, D., Schulzrinne, H.: Youslow: a performance analysis tool for adaptive bitrate video streaming. In: Proceedings of the 2014 ACM Conference on SIGCOMM, pp. 111–112, August 2014
3. Xu, X., Liu, J., Tao, X.: Mobile edge computing enhanced adaptive bitrate video delivery with joint cache and radio resource allocation. IEEE Access **5**, 16406–16415 (2017)
4. Bhat, D., Rizk, A., Zink, M., Steinmetz, R.: Network assisted content distribution for adaptive bitrate video streaming. In: Proceedings of the 8th ACM on Multimedia Systems Conference, pp. 62–75, June 2017
5. Georgopoulos, P., Elkhatib, Y., Broadbent, M., Mu, M., Race, N.: Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In: Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking, pp. 15–20, August 2013
6. Bentaleb, A., Begen, A.C., Zimmermann, R.: SDNDASH: improving QoE of HTTP adaptive streaming using software defined networking. In: Proceedings of the 24th ACM International Conference on Multimedia, pp. 1296–1305, October 2016
7. Phan-Xuan, T., Kamioka, E.: Efficiency of QoE-driven network management in adaptive streaming over HTTP. In: 2016 22nd Asia-Pacific Conference on Communications (APCC), pp. 517–522. IEEE, August 2016
8. Lederer, S.: Optimal adaptive streaming formats MPEG-DASH & HLS Segment Length, Bitmovin Inc, 9 April 2015. https://bitmovin.com/mpeg-dash-hls-segment-length/
9. "Index of/129021/dash/akamai/bbb_30fps," DASH Industry Forum. http://dash.edgesuite.net/akamai/bbb_30fps/
10. "GPAC Nightly Builds," GPAC. https://gpac.wp.imt.fr/downloads/gpac-nightly-builds/