

Evaluating HTTP Adaptive Streaming Algorithms Under Parallel TCP Connections

Saad Qaisar, Syed Hamid Rasool, and Abdul Basit(✉)

National University of Sciences and Technology,
Sector H-9 Islamabad, Islamabad, Pakistan
{saad.qaisar,14mseesrasool,basit.khan}@seecs.edu.pk

Abstract. Online video streaming services have been gaining rapid growth, and that has led to an increase in Internet traffic and have matched with traditional television video services in terms of content consumption. HTTP Adaptive streaming service has been adopted as the medium for video delivery over the Internet with its variants, HTTP Adaptive Streaming (HLS) and Dynamic Adaptive Streaming over HTTP (MPEG-DASH) supporting all playback devices.

In video streaming domain, the performance is measured in terms of the Quality of Experience (QoE) provided to the user and has been known to be influenced by the bitrate adaptation methodology. The objective scores of QoE are related to the bitrate quality delivered while keeping video stalls and rebuffering to a minimum. These include classes of buffer based and network throughput based adaptive switching mechanisms to optimize the QoE. In our work, we evaluate the performance of both these classes of algorithms over parallel HTTP connections at the client, a proven method to increase network throughput to resultantly increase the bitrate quality over an emulated WAN environment, controlled by actual traces conducted during an extensive measurement study over a distributed global DASH measurement cluster.

Keywords: Video streaming · SD-WANs · QoS

1 Introduction

Video streaming services have not stopped growing in recent years, and that has led to an increase in Internet traffic, probably due to the growth of devices with access to the network, such as phones and tablets. Today, services such as YouTube or Netflix, among others dominate much of the world as can be seen in the "Global Internet Phenomena", a report carried out by Sandvine [1] with periods of maximum download activity of up to 25% of total traffic for YouTube in Europe during July 2015. According to Akamai State of the Internet report 2016, the median connection speed in Pakistan is 2.0 Mbps as compared with a global average of 5.6 Mbps and 12.2 Mbps originating from US. Pakistan is not the only target however, still 35% of all global connections are still less than 4 Mbps. Video streaming has come a long way since its inception in the early 90s.

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11622, pp. 595–610, 2019.
https://doi.org/10.1007/978-3-030-24305-0_44

Nowadays the predominant technique for transmitting live and on-demand video is adaptive streaming using HTTP segments, the reasons that allowed it were its simple deployment and management since the HTTP infrastructure (proxies, caches and content delivery networks, CDN) was already deployed. With HTTP being also a standard protocol for the Internet.

The video streaming methods have been evolving over time, the use of the UDP protocol [11] for the delivery of live video and reliable transport protocols such as TCP [12] for the delivery of videos on demand has been decreasing. Currently, the transmission of both live video and video on demand (VoD) is done mainly through HTTP, over the TCP protocol. The main reason for the use of HTTP is the possibility of reusing its infrastructure, since it is friendly with NAT devices, which allow exchanging packets between two networks that assign mutually incompatible addresses, and also allows passing through most firewalls through port 80. Also, since the HTTP flow is directed by the client, it supports all the adaptation logic, reducing persistent server-application connections.

Adaptive Streaming is a technique used for the distribution of multimedia flow and was born as a major advance regarding the progressive discharge of the flow in order to optimize the user experience by distributing the best possible bitrate based on the user's available bandwidth final. To do this, a video is encoded in different bitrates and segmented into pieces of a few seconds, allowing each segment to be available for the different bit rates.

Dynamic Adaptive Streaming over HTTP (DASH) [14] is a standard for adaptive streaming over HTTP developed by MPEG that became an international standard in November 2011 and was published as ISO/IEC 23009-1: 2012 in April 2012 [25]. A second edition was published in May 2014 as ISO/IEC 23009-1: 2014.

The streaming model that defines MPEG-DASH offers control exclusively to the client, who requests the information through the HTTP protocol to the server [18].

The objective of our work is to improve the Quality of Experience of end users video sessions in terms of average bitrate quality and rebuffering events on real world measured traces of video streaming sessions by applying parallel HTTP connections through an emulated DASH HTTP video player.

The rest of the paper is as follows. We introduce various streaming protocols in Sect. 2 including, traditional and HTTP. In Sect. 3, we provide the system model of QoE and MPEG-DASH. In Sect. 4, we provide details on the measurement study of DASH flows and in Sect. 5, we explain the emulation tests conducted including the parallel streaming architecture. Section 6 provides discussion of the results followed by the conclusion.

2 Related Work

HTTP adaptive streaming has been an area of great interest that grew after the publication of first version of DASH ISO specifications in 2012 by the MPEG group. The loose specifications allowed the choice of adaptive streaming

algorithm to be in the hands of the video player developers to allow innovation and optimization in the bitrate adaptation controller logic.

The note worthy works using alternate approach includes the use of server-side bitrate switching [9,16,27], TCP changes to avoid bursts [2]. Our focus will be on the client-side problem for two key reasons. First, client-side solutions are immediately deployable in contrast to other solutions that require support from in-network devices [6], server-side software changes [7,28], or modifications to lower-layer transport protocols [16]. Direct interaction with content providers is also not required as some studies have been done in conjuction with the content providers live network [11,12,14,21]. Secondly, the client is often well placed to detect performance issues and respond to dynamics in network connection [22].

For the client side which is our focus, several techniques have been proposed since and several techniques and algorithms have been developed. They are largely grouped into broad classes, buffer based algorithms and bandwidth based algorithms, as they primarily influenced by either the amount of remaining buffer value or the currently observed network connection bandwidth to control the bitrate quality set [19,26]. There are also newer studies that have incorporated both these parameters and applied optimization solvers to provide the solution.

Prior works have also shown that sudden changes in available network capacity confuse existing ABR algorithms, causing the algorithms to either overestimate or underestimate the available network capacity [10,15]. The overestimation leads to unnecessary rebuffers [23]. In our work, we take separately both the throughput based algorithms as well as buffer based algorithms and measure their performance under low bandwidth conditions.

The underestimation not only fills throughput as well as buffer based approach, the buffer with video chunks of lower quality, but also leads to the ON-OFF traffic pattern in video traffic: when the playback buffer is full, the client pauses the download until there is space. In the presence of competing TCP flows, the ON-OFF pattern can trigger a bad interaction between TCP and the ABR algorithm, causing a further underestimate of capacity and a downward spiral in video quality [13]. When competing with other video players, overlapping ON-OFF periods can con-fuse capacity estimation, leading to fluctuating video quality and unfair link share among players [1]. It has also been established that the streaming algorithms that request aggressively for higher bitrates, tend to get more bandwidth out of the network.

A related work uses parallel connections to download video files, which were not in DASH format, in different representations as a whole and use the download time improvement to claim better performance [3]. However, the video downloaded was not segmented and hence, the video quality could not be adapted. Another study downloaded segments in parallel [24] and while their they can be applied to DASH streaming logic, their work fetched segments in parallel from different sections in the video to aid seeking and skipping ahead in videos using some look ahead content that is pre-downloaded using their parallel technique. The last similar study was performed through parallel streaming using

multiple server technique in which the same video content was placed on different servers. Their technique is called MD-DASH (multiple descriptor DASH) [7] and requires customization to the original DASH manifest file format and is hence, non-compliant with the ISO-BMFF standard.

To the best of our knowledge, parallel video streaming using the standard DASH architecture has not been tested, evaluated or compared with the existing state of the art adaptive bitrate switching algorithm to measure the performance gains that can be achieved for video streaming sessions.

3 System Model

Video quality is a key benchmark for any streaming service, be it traditional broadcast, cassettes, DVDs or Internet video. It has been evaluated by various subjective and objective metrics. We provide the models for the currently used metrics for HTTP streaming after describing all the options. We will also model the typical MPEG-DASH playback model after its protocol architecture.

3.1 DASH Model

We will now model the overall MPEG-DASH system architecture [20]. In order to enable DASH streaming, the video file has to be encoded in multiple bit rates and resolutions and then each of the file is then segmented into small chunks. As per DASH specification, the segment size can be set from 2–10 s.

Here is the abstract model of the DASH video player in Fig. 1.

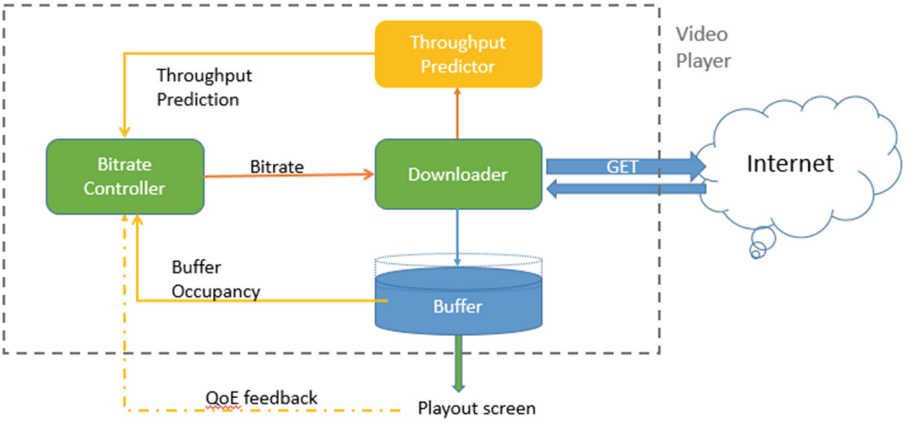


Fig. 1. Abstract model of DASH player.

On the client side, the selection logic is performed by the bitrate controller [25]. The controller takes into account, the feedback of the network throughput and amount of video content stored in buffer and makes the decision by setting the bitrate of the next segment after taking into account the history of previous segment rates. This logic decoupled from the main MPEG-DASH protocol fixed standard, i.e. the ISO-BMFF format [20] of the manifest (mpd) file of the DASH encoded video and the playback engine, allows to develop custom framework in order to improve video streaming performance by researchers and video service owners.

3.2 DASH Specifications

When the video is prepared into DASH format, the final produced highest resolution video (mezannine file) is split into segments (chunks) of 2–10 s of duration. This number is consistent for a DASH video. The next step is the transcoding of video into multiple representations with a set bitrate and resolution (e.g. 1080p at 3000 kbps, 720p at 2000 kbps, 240p at 300 kbps). The complete metadata of the file: duration of the video; duration of segment; file path of each segment.

The MPD file is the strict standard which is a component of all MPEG-DASH video. In the below Fig. 2, the complete architecture of DASH is presented.

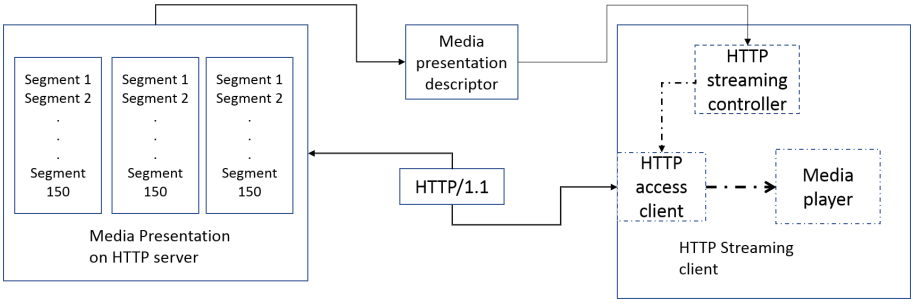


Fig. 2. MPEG-DASH architecture.

In the above diagram, the specifications of DASH client and DASH web server are detailed. The important aspect of the diagram is that the solid blocks are required to be fixed as per standard, while the dotted blocks are up to the implementation of DASH by the users. The fixed blocks include the segmentation and transcoding of file in different representations as shown in the segment blocks as well the descriptor file in accordance. The dotted blocks represent the bitrate

controller, HTTP client (directed by the streaming controller) and the media player. This enables both innovation in terms of engineering as well as design as each DASH client is visually different from each other as presented in YouTube, Netflix as well as most of the other large media content providers on the Internet since DASH is now widely adopted.

4 Protocol Details

In our study, we apply a unique approach to improve DASH performance through establishing multiple HTTP connections to the video server fetching data in parallel. The fetching of data in parallel has been useful for multiple applications running over the Internet - the Internet Download Manager (IDM) uses this technique to speed file downloads while modern FTP clients, like FileZilla also use them in order to perform data transfers. This parallel technique is visualized below (Fig. 3).

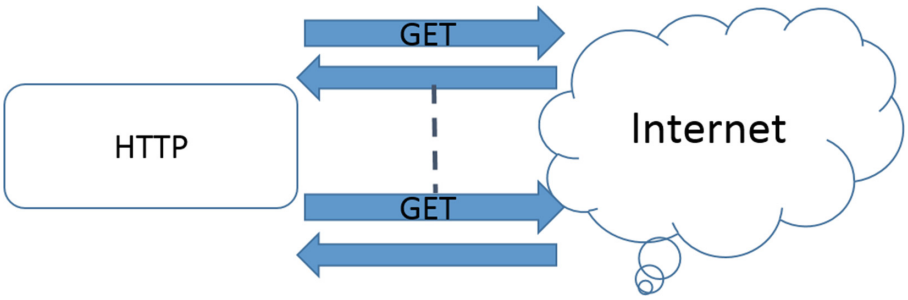


Fig. 3. Parallel HTTP connections.

4.1 Parallel Streaming Algorithm

After establishing parallel connections by setting a predefined number, we proceeded to create a dynamic algorithm for this purpose. We use the information provided in the DASH manifest file to obtain the size of all segments for various resolutions and tune the throughput based adaptive bitrate algorithm to suit for dynamic parallel HTTP throughput based algorithm.

Parallel ABR algorithm
Selection of the batch size, T (default T=2)
Initialization of T by obtaining size of the segment z (in bytes) through HTTP HEAD request.
z[n] is less than β_L
T[n] = 1single connection
z[n] is greater than β_H
T[n] = 88 parallel connections
z[n] is greater than 1.15z[n-1]
T[n] = min(T[n-1]++,5)increase connections
z[n] is less than 0.85z[n-1]
T[n] = max(T[n-1]–,1)decrease connections

Select next segment rate, *R[n]*
1) Estimate bandwidth by equating to the measured throughput

$$x'[n] = x[n-1]$$

2) Calculate the moving average of the last K segments

$$y = \sum_{k=1}^{K} x_k[n]$$

3) Select R' by calculating the weighted average

$$R'[n] = \gamma x'[n].(1-\gamma)y$$

4) Quantize R' to the the set of available bitrates to determine the next rate

$$R[n] = Q(R'[n])$$

Initially after running controlled experiments with parallel connections we observed that, for some reason we were not able to gain any performance advantage through varying the amount of parallel connections dynamically as opposed to setting a fix number of parallel connections. Upon careful observation of the dynamics of HTTP parallel connections, we found that when sequential one-by-one segment download occurs, we can track the download time of each segment (and hence the download rate) and predict the best bitrate for the next one. This approach works well for the state of art throughput based algorithms, but we cannot use it with parallel downloads, unfortunately.

For simplification we can think that every batch will be downloaded in batch_size parallel connections. We use last batch download size, last batch download time and average size (in bytes) of last batch segments. The algorithm then adjusts the batch size after the each batch download to increase next average batch download rate. Once this is set, it is incorporated into the main throughput based algorithm which selects the next bitrate. Our parallel algorithm determines the batch_size for the next segment. It is interesting to note that although our parallel algorithm greatly affects the results of the throughput

based controller, it is completely transparent and does not affect the algorithm, instead the performance is improved due to the resulting improvement in bandwidth which is used in the calculations.

5 Measurement Study

The motivation of our work is to improve the video QoE for lower bandwidth regions. For this purpose, we performed extensive measurements to calculate the video quality as experienced by endpoints in a third world country with poor overall Internet speed, as evident from stats in the Sandvine global Internet report.

5.1 Neubot

Neubot is an open source tool for network collection and measurement. The operation of Neubot is that when the software is installed on the client, it automatically runs several tests after each 30 min interval: download speedtest, upload speedtest, bittorrent test and dashtest [4]. It connects to the central neubot server where it is redirected to one of 124 server placed in one of 40 different geographical locations across the world and one of the four types of tests at random. Since we are interested in measuring video streaming performance, we modified the software actions to enforce it to run dashtest only after every 20 min through a Python script running continuously in the Windows background for a number of months.

5.2 Measurement Setup

The study was conducted on two different Internet providers. Since we had to minimize affect of externalities, we conducted all experiments on wired connections. First one was performed over the educational backbone Internet provider, PERN. The second variant was performed over the home connection from a large fixed broadband ISP located in Islamabad.

5.3 Measurement Results

Overall, we conducted almost 20,000 separate video sessions during the period from January to July 2017. Out of these, 14000 came from the backbone network and Here are some of the notable overall stats of the data collected. The below Table 1 shows the key statistical figures from the two settings.

Table 1. Statistical overview of the measurement study.

ISP	Backbone	Commercial	Combined
No of video sessions	13789	6029	19818
No of downloaded segments	210073	91462	301535
No of different servers	86	119	119
No of rebuffering events	2483	846	3372
No of incomplete sessions	574	140	714
Avg download speed (kbps)	1046	1136	1074
Avg bitrate quality (kbps)	803	816	807

5.4 Analysis

The following is the distribution of the frequency plot that we obtained for the readings. It is evident that for single HTTP connections in low bandwidth setting, the throughput is poor relative to the capacity of the network link. This behavior is also a result of less aggressive approach in selection of bitrate as discussed in a previous study (Fig. 4).

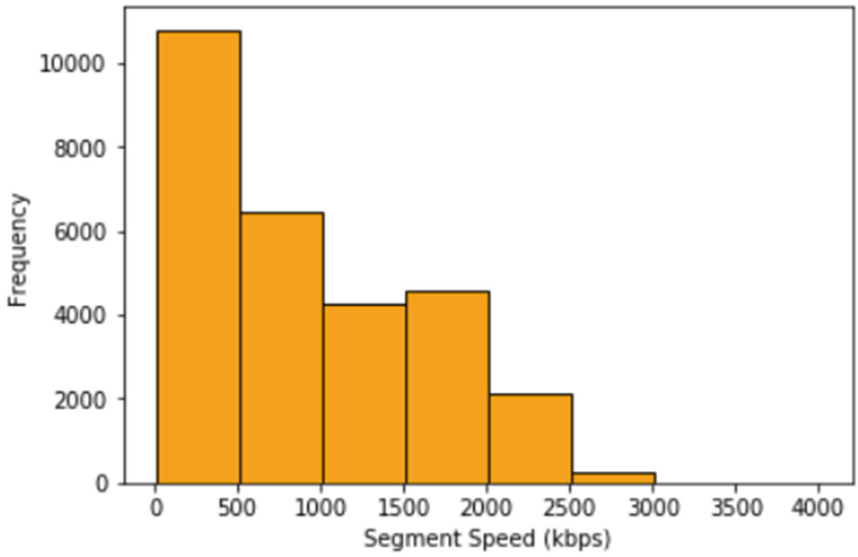


Fig. 4. Frequency plot of segment download speeds.

We also observed a high amount of rebuffering events that occured over the course of our study, over 3000 sessions experienced rebuffering events while over 700 sessions failed to complete due to network failure during the playback

process. Below is a plot that represents the number of buffering events over the course of 24 h period. The average download time for every segment was 1.6 s which is lower than 2 which meant that a large proportion of video sessions were smoothly played out.

The last and perhaps the most significant observation from the statistics were the average download speed and average bitrate quality. Considering that the two Internet providers had 3 Mbps and 4 Mbps links and we conducted the experiments throughout the 24 h mark to neutralize any effect on any background traffic activity that might affect the application performance. The average speeds determined were a meager 1046 kbps and 1136 kbps for the backbone and commercial ISP respectively. This leaves a huge gap for potential improvements through parallel connections as only a third of the capacity is being utilized on average.

6 Video Streaming Experiment

After evaluating the state of Internet video performance, we proceeded to conduct controlled experiments to improve the Quality of Experience (Fig. 5).

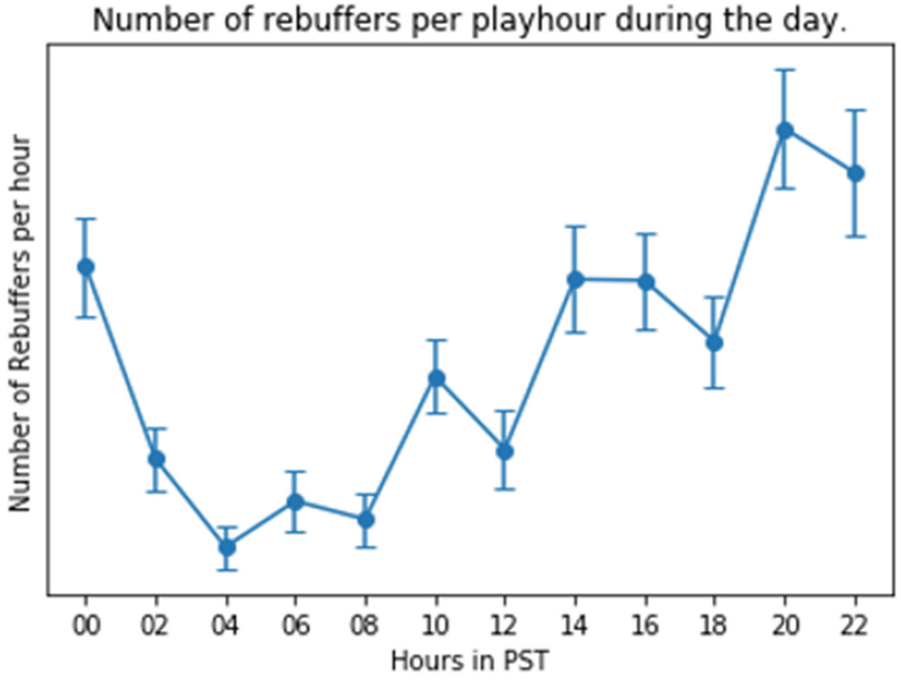


Fig. 5. Rebuffering events occurrance per hour.

6.1 DASH Encoding

We start by encoding the standard Big Buck Bunny video into 5 different representations. The process is performed by an open source utility FFMpeg which is available for both Linux and Windows platforms. The following Table 2 represents the resolutions and bitrates to which we chunked and encoded into the ISO-BMFF format of the DASH manifest file.

Table 2. Video representations.

Resolution	360 × 240	480 × 360	640 × 480	1024 × 720	1920 × 1080
Bitrate (kbps)	350	600	1000	2000	3000

6.2 Experiment Setup

To conduct testing of our parallel algorithm, we used an open source DASH emulator framework named AStream. The player framework is built using Python and uses various HTTP libraries to download and parse through the DASH manifest file to obtain all resolution and bitrate information. The player then emulates playback by downloading the video segments and selecting the next bitrate through 1 of the 2 state of the art throughput based and a buffer based adaptive switching algorithm.

The client and server instances are virtual machines placed and orchestrated on a centrally controlled academic cloud, named GENI. The GENI project [5] enables research studies with the facility of spinning up Linux and BSD instances and allows them to remotely access the machines to run experiments and execute scripts. The same topology can be replicated on a single PC using a hypervisor, such as KVM, VirtualBox or VMWare Player to connect 2 VMs. The setup can also be created by connecting 2 physical computers or servers together by LAN cable or even through networking of 2 docker containers running on Linux host Operating system.

The server is running Apache web server and the DASH encoded Big Buck Bunny video, a common evaluation video [17], segments have been placed on it along with the manifest file. On the client side, a Net shaper module has been added to emulate a WAN connection between the client and server so that the network conditions obtained through our measurement study is replicated. Dummynet [8] is used to control the latency and bandwidth of the emulated WAN connection. From literature, the ping latency is set to 80 ms while the throughput is taken through random samples from the 19000 data points of our study.

6.3 Parallel Streaming Results

We modify the code in the AStream emulator to take advantage of a system utility named aria2 which facilitates provision for parallel HTTP downloading

according to our parallel streaming algorithm explained in Sect. 3. We integrate aria2 functions that split the file into multiple chunks and fetch them in parallel.

The following results were obtained when we applied our algorithm in the single connection scenario (default state of the art), and with 2, 3 and 4 parallel connections as labelled by N equals 1, 2, 3, 4 in the figure. We observed that due to being small files, there was a loss of improvement after increasing the connections greater than 4 as the segments are further split into smaller files causing greater overhead than the performance advantage gain.

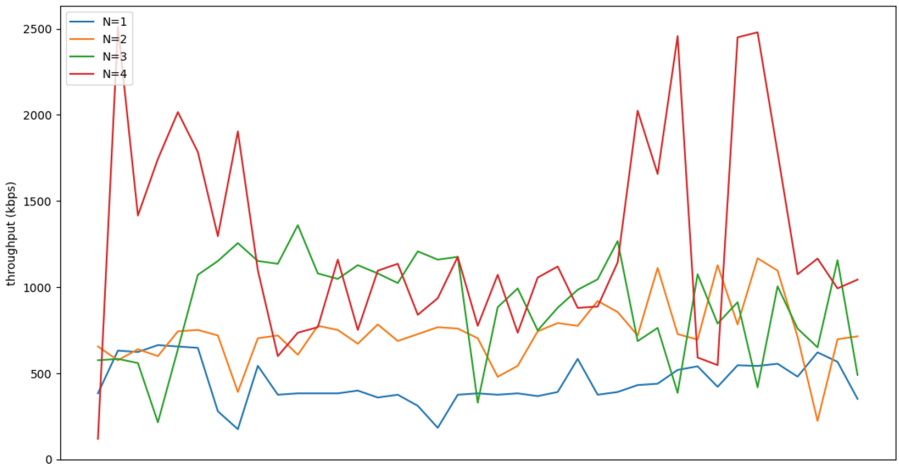


Fig. 6. Download speed of parallel video streaming.

The results show that parallel HTTP connections provide a considerable performance improvements over the single HTTP connection when evaluated for a state of the art throughput based ABR algorithm. The trend observed is that the throughput generally increases with increasing the number of parallel connections. Some interesting observations are made from the plot, when higher bitrates are selected, the file size becomes larger and the parallel connections tend to benefit further as seen from the peaks in case of 3 and 4 parallel connections.

The result of multiple runs in terms of the average throughput segment downloading speed has improved to more than double as in the case of the default single HTTP connection in case of the same throughput based ABR algorithm (Fig. 6).

The following Fig. 7 represents the bitrate quality selected for both single and dynamic parallel connections for both throughput based and buffer based algorithms.

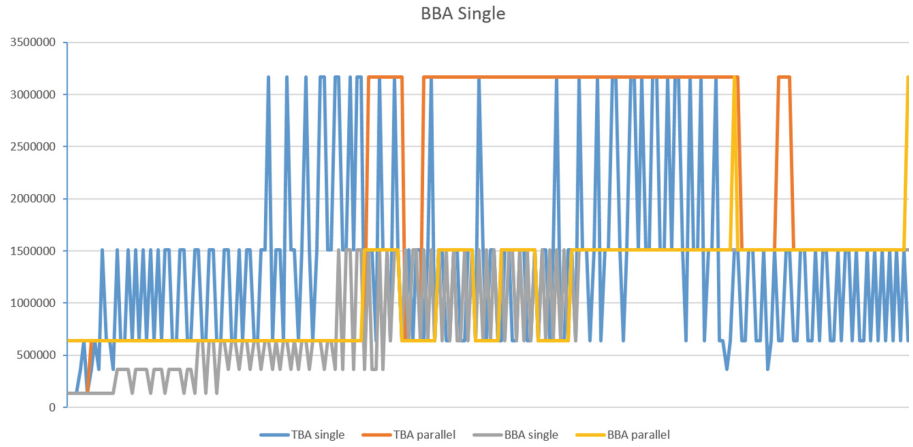


Fig. 7. Bitrate quality of parallel video streaming.

For the same experiment, the below bar chart represents the average speed during the runs to show the clear difference between the video quality improvement (Fig. 8).

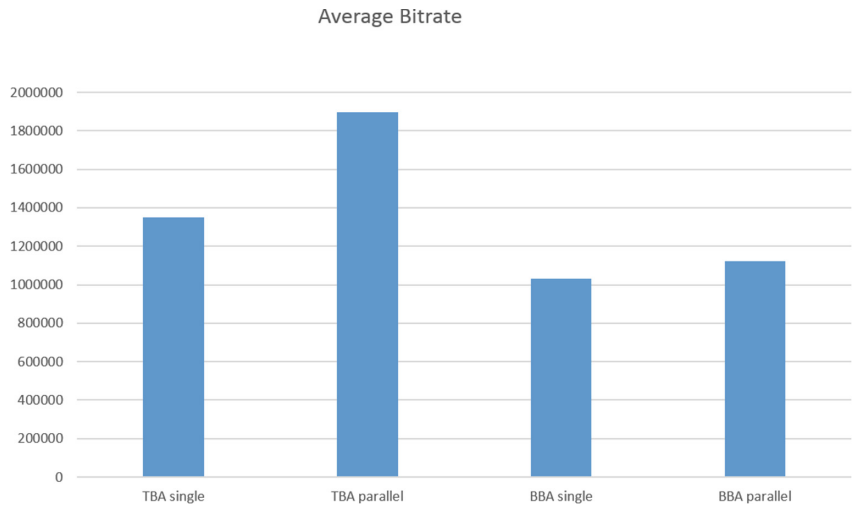


Fig. 8. Bitrate quality of parallel video streaming.

6.4 Simultaneous Running

We also checked the performance comparison while conducting 2 simultaneous video streaming sessions. An instance of the player with single connection and a second instance with a parallel dynamic option were started at the same time. The following graph shows the average bitrate quality:

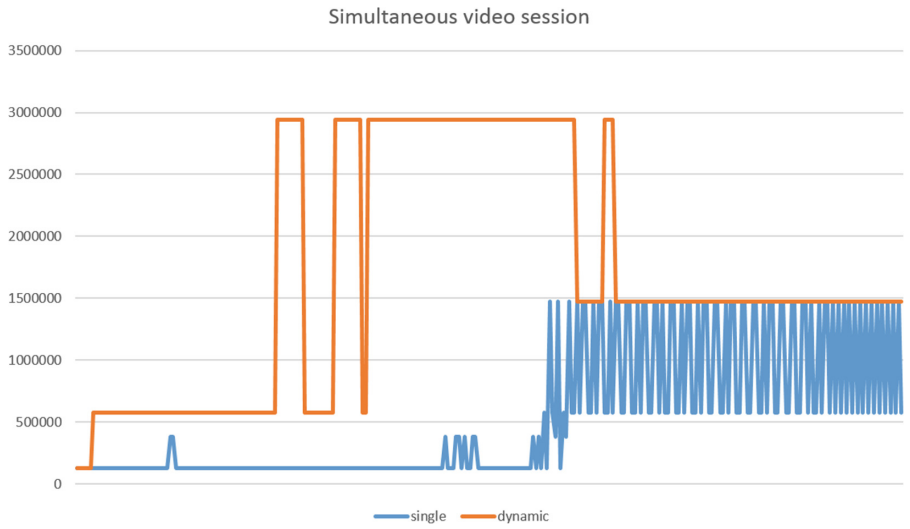


Fig. 9. Bitrate quality of parallel video streaming.

In comparison to the original scheme, the dynamic parallel connection drastically improves the performance at the expense of the other flow, which is the single connection (Fig. 9).

7 Conclusion

This work was motivated by users frustrations of poor video streaming experience in developing countries where the average Internet bandwidth connection is fairly constrained. The global average Internet connection speed is 5.6 Mbps compared to 2.0 Mbps in Pakistan.

Our extensive measurement study involved a video client establishing video sessions with 119 servers located in different geographical locations over a course of 6 months and collecting statistics of almost 20,000 video sessions out of which some of them were prone to buffering events and some even failure.

As we found out the margins for improvement through our measurement study, our test bed results validated our thoughts after observing up to twice the segment downloading throughput rates as a result of utilizing multiple connections for the same state of the art adaptive streaming algorithm.

References

1. Akhshabi, S., Anantakrishnan, L., Begen, A.C., Dovrolis, C.: What happens when HTTP adaptive streaming players compete for bandwidth? In: Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video, pp. 9–14. ACM (2012)
2. Ameur, C.B.: TCP protocol optimization for HTTP adaptive streaming. Ph.D. thesis, Rennes 1 (2015)
3. Ansari, M., Ghaderi, M.: Parallel HTTP for video streaming in wireless networks. In: IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 337–342. IEEE (2016)
4. Basso, S., Servetti, A., Masala, E., De Martin, J.C.: Measuring dash streaming performance from the end users perspective using neubot. In: Proceedings of the 5th ACM Multimedia Systems Conference, pp. 1–6. ACM (2014)
5. Berman, M., et al.: GENI: a federated testbed for innovative network experiments. Comput. Netw. **61**, 5–23 (2014)
6. Bhat, D., Rizk, A., Zink, M., Steinmetz, R.: SABR: network-assisted content distribution for QoE-driven ABR video streaming. ACM Trans. Multimed. Comput. Commun. Appl. (TOMM) **14**(2s), 32 (2018)
7. Bruneau-Queyreix, J., Negru, D., Batalla, J.M., Borcoci, E.: Multiple description-DASH: pragmatic video streaming maximizing end-users' quality of experience. In: IEEE International Conference on Communications (ICC), pp. 1–7. IEEE (2016)
8. Carbone, M., Rizzo, L.: Dummynet revisited. ACM SIGCOMM Comput. Commun. Rev. **40**(2), 12–20 (2010)
9. Concolato, C., Le Feuvre, J., Denoual, F., Nassor, E., Ouedraogo, N., Taquet, J.: Adaptive streaming of HEVC tiled videos using MPEG-DASH. IEEE Trans. Circuits Syst. Video Technol. **28**, 1981–1992 (2017)
10. De Cicco, L., Caldaralo, V., Palmisano, V., Mascolo, S.: Elastic: a client-side controller for dynamic adaptive streaming over HTTP (DASH). In: 20th International Packet Video Workshop (PV), pp. 1–8. IEEE (2013)
11. Hoßfeld, T., Schatz, R., Biersack, E., Plissonneau, L.: Internet video delivery in YouTube: from traffic measurements to quality of experience. In: Biersack, E., Callegari, C., Matijasevic, M. (eds.) Data Traffic Monitoring and Analysis. LNCS, vol. 7754, pp. 264–301. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36784-7_11
12. Hoßfeld, T., Seufert, M., Sieber, C., Zinner, T., Tran-Gia, P.: Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies. Comput. Netw. **81**, 320–332 (2015)
13. Huang, T.-Y., Handigol, N., Heller, B., McKeown, N., Johari, R.: Confused, timid, and unstable: picking a video streaming rate is hard. In: Proceedings of the 2012 Internet Measurement Conference, pp. 225–238. ACM (2012)
14. Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., Watson, M.: A buffer-based approach to rate adaptation: evidence from a large video streaming service. ACM SIGCOMM Comput. Commun. Rev. **44**(4), 187–198 (2015)
15. Jiang, J., Sekar, V., Zhang, H.: Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. IEEE/ACM Trans. Networking (TON) **22**(1), 326–340 (2014)
16. Kua, J., Armitage, G., Branch, P.: A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. IEEE Commun. Surv. Tutor. **19**(3), 1842–1866 (2017)

17. Lederer, S., Mueller, C., Timmerer, C., Concolato, C., Le Feuvre, J., Fliegel, K.: Distributed DASH dataset. In: Proceedings of the 4th ACM Multimedia Systems Conference, pp. 131–135. ACM (2013)
18. Lederer, S., Müller, C., Timmerer, C.: Dynamic adaptive streaming over HTTP dataset. In: Proceedings of the 3rd Multimedia Systems Conference, pp. 89–94. ACM (2012)
19. Li, Z., et al.: Probe and adapt: rate adaptation for HTTP video streaming at scale. IEEE J. Sel. Areas Commun. **32**(4), 719–733 (2014)
20. Mpeg, I.: Information technology-dynamic adaptive streaming over HTTP (DASH)-part 1: media presentation description and segment formats. Technical report, ISO/IEC MPEG (2012)
21. Plissonneau, L., Biersack, E.: A longitudinal view of HTTP video streaming performance. In: Proceedings of the 3rd Multimedia Systems Conference, pp. 203–214. ACM (2012)
22. Rainer, B., Lederer, S., Müller, C., Timmerer, C.: A seamless web integration of adaptive HTTP streaming. In: Proceedings of the 20th European Signal Processing Conference (EUSIPCO), pp. 1519–1523. IEEE (2012)
23. Sieber, C., Blenk, A., Hinteregger, M., Kellerer, W.: The cost of aggressive HTTP adaptive streaming: quantifying YouTube's redundant traffic. In: IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 1261–1267. IEEE (2015)
24. Smanchat, S., Sangkul, K., Tham, J.Y.: Enabling parallel streaming of multiple video sections by segment scheduling. In: Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia, pp. 221–226. ACM (2015)
25. Stockhammer, T.: Dynamic adaptive streaming over HTTP-: standards and design principles. In: Proceedings of the Second Annual ACM Conference on Multimedia Systems, pp. 133–144. ACM (2011)
26. Wang, C., Rizk, A., Zink, M.: SQUAD: a spectrum-based quality adaptation for dynamic adaptive streaming over HTTP. In: Proceedings of the 7th International Conference on Multimedia Systems, p. 1. ACM (2016)
27. Zhang, S., Li, B., Li, B.: Presto: towards fair and efficient HTTP adaptive streaming from multiple servers. In: IEEE International Conference on Communications (ICC), pp. 6849–6854. IEEE (2015)
28. Zhou, C., Lin, C.-W., Guo, Z.: mDASH: a Markov decision-based rate adaptation approach for dynamic HTTP streaming. IEEE Trans. Multimedia **18**(4), 738–751 (2016)