Parallel HTTP for Video Streaming in Wireless Networks

Mohsen Ansari
Dept. of Computer Science
University of Calgary
Calgary, Canada
mohsen.ansari@ucalgary.ca

Majid Ghaderi
Dept. of Computer Science
University of Calgary
Calgary, Canada
mghaderi@ucalgary.ca

***Abstract*—To stream video using HTTP, a client device sequentially requests and receives chunks of the video file from the server over a TCP connection. It is well-known that TCP performs poorly in networks with high latency and packet loss such as wireless networks. On mobile devices, in particular, using a single TCP connection for video streaming is not efficient, and thus, the user may not receive the highest video quality possible. In this paper, we design and analyze a system called *ParS* that uses parallel TCP connections to stream video on mobile devices. Our system uses parallel connections to fetch each chunk of the video file using HTTP range requests. We present measurement results to characterize the performance of ParS under various network conditions in terms of latency, loss rate and bandwidth.**

***Index Terms*—parallel HTTP; Video streaming; DASH;**

I. Introduction

A. Motivation

Video streaming over HTTP has become extremely popular and is adopted by major online streaming services such as YouTube and Netflix. Internet video streaming now takes the majority of worldwide Internet traffic and its share will continue to grow. It is estimated that, globally, Internet video traffic will be 80 percent of all consumer Internet traffic in 2019, up from 64 percent in 2014 [1].

There are several HTTP-based video streaming services implemented by different organizations. Adobe's HTTP Dynamic Streaming [2], Apple's HTTP Live Streaming [3] and Microsoft's Smooth Streaming [4] are a few examples of such services. Meanwhile, Dynamic Adaptive Streaming over HTTP (DASH) is being developed as an international standard to unify HTTP-based video streaming over the Internet [5].

The DASH standard is composed of two main parts. One part defines the Media Presentation Description (MPD) that is used by the client to discover the URLs for accessing the video content. The other part defines the format of the video content. In a DASH system, the video streaming server sends video content to clients via the HTTP protocol. Before transmission, a video is encoded into different bit rates on the server. The encoded video file at every bit rate is fragmented into small *chunks*, each chunk contains only several seconds (*e.g.*, 10 seconds) of the video. A client sends HTTP requests to the server to download video chunks sequentially. At the time of streaming, a client can dynamically change the target video chunk's bit rate based on its available resources such as available bandwidth and remaining battery [6].

HTTP video streaming systems, including DASH, rely on TCP which has poor performance in networks with high latency and packet loss. In such networks, TCP and consequently the video streaming application is unable to fully utilize the available bandwidth leading to lower quality of experience for users [7]. We measured the measured throughput of a single TCP connection[1] over a 10 Mbps link. We increase round-trip-time (RTT) under different packet loss rates and measure the throughput. When RTT and packet loss are set to 50 milliseconds and 0.1%, respectively, TCP can utilize nearly 86% of the available bandwidth. By increasing RTT and packet loss the throughput drops drastically to a point where, when RTT is set to 150 milliseconds and packet loss is 1%, TCP can only utilize 13% of the bandwidth.

Wireless networks often suffer from high latency and packet loss [8]. Moreover, wireless bandwidth is generally more limited compared to wired bandwidth. This means that HTTP-based video streaming over mobile devices is negatively affected by TCP's inability to fully utilize the wireless bandwidth, which could result in lower video quality and buffering delays during video playback. To improve TCP throughput, the use of parallel connections has been considered. For example, GridFTP [9] is an extension of traditional FTP in which multiple TCP connections are used to transfer data resulting in significant improvements over single connection FTP. This concept has been also applied to video streaming. For instance, Parallel TCP connections have been used in [10] and [7] to improve the quality of video by using multiple connections to download multiple video chunks from the server simultaneously.

B. Our Work

In this work, we also employ parallel TCP connections to improve video streaming quality on mobile devices. However, different from the existing works (as in [10] and [7]), rather than using multiple connections to fetch different video chunks, we use multiple connections to download the *same chunk*. We note that when multiple connections are used, the

[1]We use the terms "flow" and "connection" interchangeably.

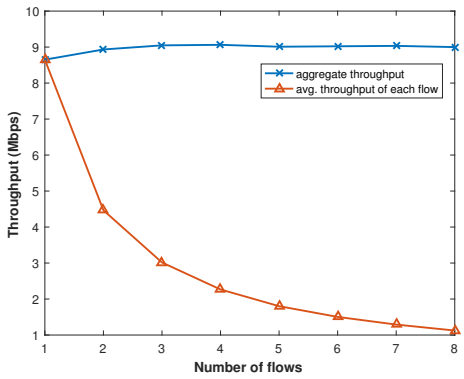


Fig. 1: Parallel TCP connections with loss probability of 0.1% and RTT of 50 ms. While the aggregate throughput increases with the number of connections, the per-connection throughput decreases.

available bandwidth between the client and server is divided among the connections. This means that, while the aggregate throughput increases, the throughput of each individual connection actually *decreases* as depicted in Fig. 1. For instance, when there are 5 parallel connections the throughput of each connection on average is less than 2 Mbps.

In wired networks, where the bandwidth between the client and server is plenty, even the reduced per-connection bandwidth is large enough to sustain video streaming at a good quality. Thus, it makes sense to use multiple connections to download different video chunks. In wireless networks however, the bandwidth is limited. Thus, when using multiple connections, the share of each individual connection may be so low that it cannot sustain video playback at a reasonable quality resulting in playback stalls. This problem happens whenever a chunk is needed *as soon as possible* to avoid a playback stall. For example, consider the first chunk of the video. It takes 3 *times longer* to download the first chunk and start the video when using parallel connections to download different chunks (*i.e.*, existing works) compared to downloading the same chunk (*i.e.*, our work). In wireless networks, when the available bandwidth fluctuates significantly over time, this situation (*i.e.*, needing chunks quickly to avoid stalls) would be even more likely to happen compared to wired networks.

The work presented in this paper has two parts. In the first part, we focus on using multiple connections for video streaming as described above. We design and evaluate *Parallel Streaming* (ParS) that employs parallel HTTP for video streaming in low bandwidth and high loss networks. In the second part, we turn our attention to determining the relationship between the aggregate throughput and the number of parallel TCP connections. As discussed later, a critical problem in all systems that use parallel TCP connections (including the above mentioned works) is to decide how many connections are needed to achieve the best video playback quality. We show that as the number of connections increases, the aggregate throughput also increases up to a certain point. Beyond this point, increasing the number of connections actually results in lower throughput due to increased competition among the connections and increased processing and computation overhead.

Our contributions in this paper can be summarized as follows:

- We develop a protocol to use parallel TCP connections to download each chunk of a video file from the server using HTTP range requests.
- We develop a prototype system based on our protocol called Parallel Streaming (ParS) and deploy it on a testbed.
- We then conduct measurement experiments to analyze the impact of RTT, packet loss, bandwidth and chunk size on the throughput performance of our protocol.

C. Related Work

There is a vast amount of literature on video streaming (please refer to [11] for a recent survey). The following works are more specific to our work which is on the use of parallel HTTP for enhancing the quality of video streaming.

1) Parallel TCP for File Transfer: One of the main application areas of parallel TCP connections is in file transfer. GridFTP [9] uses parallel TCP to improve the performance of File Transfer Protocol (FTP). Many download manager softwares such as Internet Download Manager (IDM) [12] and Download Accelerator Plus (DAP) [13] use this concept for increasing download speed.

2) Parallel TCP for Video Streaming: In the area of HTTP streaming, the authors in [10] used parallel HTTP connections and suggested to gradually increase the number of connections based on the available bandwidth and network conditions. Their approach is based on initiating a separate connection for each video chunk. The work in [14] uses content centric networking (CCN) and multiple network interfaces typically available on a mobile device to implement parallel streaming. It uses multiple connections to download the same video content. However, at any point in time, it uses the connection which delivers data at the highest rate. The work in [7] uses multiple connections to reduce throughput fluctuations in networks with high packet loss and RTT. It uses 10 parallel connections to overcome the poor performance of TCP in such networks, though the main focus is to maintain fairness among connections with favorable and poor network conditions in terms of packet loss and RTT. The authors in [15] use multiple TCP connections to enable users to play different parts of the video without having to wait for buffering of the entire video. Finally, [16] uses multiple connections to download different layers of the video with scalable video coding.

D. Paper Organization

The rest of the paper is organized as follows. Section II is dedicated to presenting our approach and includes measurement results obtained from experiments on our testbed. A summary of our measurement results is presented in Section III. Section IV concludes the paper.

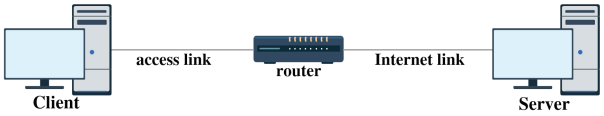


Fig. 2: Network topology used for measurements and simulations.

II. ParS: Parallel Streaming

A. overview

The problem with existing work is that, multiple connections are used to download distinct segments of the video file in parallel. This approach will improve the overall download time and throughput specially when the bottleneck link is in the core of the network. To see this, consider the network topology depicted in Fig. 2, where the link between the router and server is assumed to be the bottleneck link. Assume the bottleneck link has capacity C. If the current number of TCP flows through the link is M and the client starts N parallel TCP connections, then the aggregate throughput denoted by X_N is approximately given by:

$$X_N = \frac{N}{N+M}C, \tag{1}$$

which is due to fairness property of TCP. Clearly, we are assuming a homogeneous case where all flows have similar characteristics (*e.g.*, same round trip time). Notice that, in this case, X_N is an increasing function of N. However, as N increases, the increase in X_N becomes negligible. Assume measurements for X_1 and X_2 are available. It is obtained that,

$$X_1 = \frac{1}{1+M}C, \qquad X_2 = \frac{2}{2+M}C, \tag{2}$$

which can be used to obtain the following estimates for M and C,

$$C = \frac{X_1 X_2}{2X_1 - X_2}, \qquad M = \frac{2(X_2 - X_1)}{2X_1 - X_2}. \tag{3}$$

Two observations are made about the behavior of X_N:

- If $N \gg M$ then, $X_N \approx C$.
- If $N \ll M$ then, $X_N \approx NX_1$.

Fig. 3 shows the throughput achieved with different number of background and parallel TCP flows, when the capacity of the bottleneck Internet link is set to 10 Mbps, 50 ms RTT and 0.1% packet loss. As can be seen, by increasing the number of parallel connections, the aggregate throughput increases. Now, let us consider the per-connection throughput. From (1) we have,

$$\frac{X_N}{N} = \frac{C}{N+M}, \tag{4}$$

which indicates that as the number of parallel connections N increases, the per-connection throughput decreases. As mentioned earlier, there are two competing effects in play:

1) **Aggregate throughput:** Aggregate throughput can be increased by using parallel connections. This is specially true when the bottleneck link is shared by other background traffic (*e.g.*, the link is at the core of the network). In this case, increasing the number of parallel connections effectively *steals* bandwidth from the background traffic. If the link is not shared with other users (*i.e.*, no background traffic) then using multiple connections helps improve the link utilization.
2) **Per-connection throughput:** Per-connection throughput decreases as the number of parallel connections increases. If the bandwidth of the bottleneck link is sufficiently high then the share of each connection of the bandwidth could still be high enough to support smooth video streaming at high quality. However, if the capacity of the bottleneck link is low, when each connection is downloading a distinct video chunk, this approach may lead to playback stalls depending on the level of synchronization among the connections (more on this later in the paper).

When streaming video on mobile devices, the low bandwidth wireless access link is usually the bottleneck. In this case, using multiple connections to download separate video chunks increases the download time of some video chunks. As discussed in the Introduction section, these are the chunks that need to be downloaded as fast as possible to avoid stalls. Our solution to this problem is to have multiple TCP connections downloading one video chunk at a time. This way, the main focus of our approach is to download the next chunk as soon as possible. This decreases the chance of stalls in video playback and speed ups the playback startup. Fig. 4 depicts the difference between existing approaches and our proposed approach (ParS) when using two parallel connections.

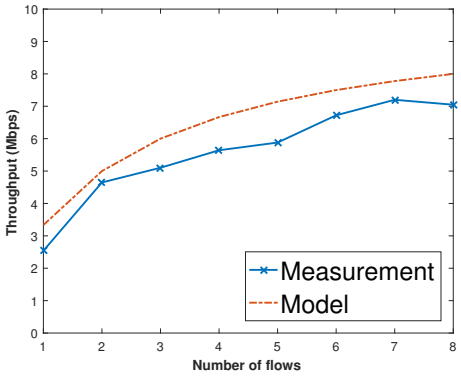
B. System Design

With ParS, to fetch a chunk, the client first obtains information about the size of the chunk by sending an HTTP-HEAD request to the streaming server[2]. Then it divides the chunk size by the number of parallel connections to find the range of each request. The client then sends multiple HTTP-GET requests each requesting a different byte-range (using *Range* attribute in HTTP request header) of the chunk, where each request is sent over a separate TCP connection. Then the client waits for the streaming server to send the chunk parts in HTTP responses. As the chunk parts arrive at the client, it reconstructs the original chunk. Algorithm 1 illustrates the steps for downloading a video chunk in ParS. This algorithm is repeated for each chunk sequentially.

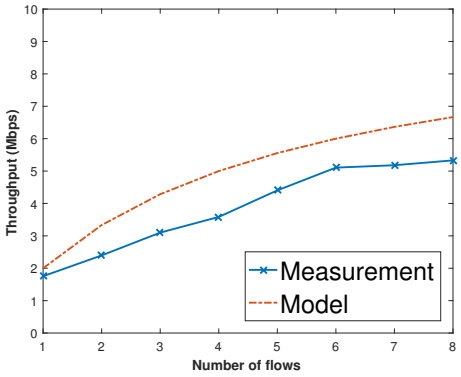
C. Testbed Specification

To study the effectiveness of our approach, we built a simple testbed consisting of a client and server running Ubuntu 14.04 LTS (with 8 GB RAM and Dual core 2.4 GHz CPU) connected via a D-link switch with 1 Gbps Ethernet cables. All TCP related parameters (*e.g.*, socket buffer size) are the default values in the Linux kernel. We use wired links in

[2]This step could be eliminated by embedding the size information in the MPD file when using DASH.

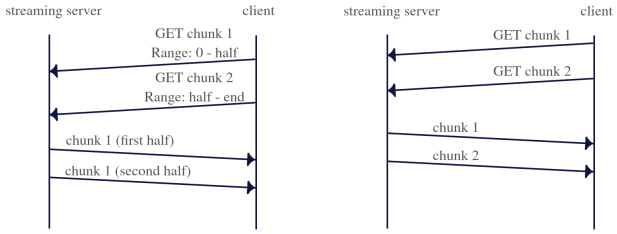


(a) $M = 2$ background flows.



(b) $M = 4$ background flows.

Fig. 3: Aggregate TCP throughput with increasing number of parallel connections obtained from the measurements and model. The Internet bottleneck link has 10 Mbps bandwidth.



(a) ParS approach. (b) Existing approach.

Fig. 4: Video streaming using two parallel connections.

Algorithm 1 Parallel Streaming (ParS)

1: $n \leftarrow$ number of parallel TCP connections
2: Send `HTTP-HEAD` request for the chunk
3: $chunksize \leftarrow$ chunk size from `HTTP-HEAD` response
4: $range \leftarrow chunksize/n$
5: **for** $i = 0$ **to** $n - 1$ **do**
6: $start \leftarrow i \times range$
7: $end \leftarrow (i + 1) \times range$
8: Send `HTTP-GET` with `Range` set to $[start, end]$
9: **end for**
10: **for** $i = 0$ **to** $n - 1$ **do**
11: Receive HTTP response
12: Store partial chunk in buffer
13: **end for**
14: Restore the chunk

order to have the ability to accurately control RTT, packet loss and bandwidth without any random fluctuations due to wireless interference or collisions. However, various network parameters in our experiments (such as bandwidth, packet loss, and RTT) are set so that we create a network behavior that is close to what would be observed in wireless networks.

To achieve this, we used *tc* and *netem* applications for simulating packet loss, RTT, and controlling transmission rate. Traffic control (*tc*) is part of the Linux kernel which allows the user to access networking features. It has three main features: monitoring the system, traffic classification, and traffic manipulation. The utility *netem* is an extension of *tc*, which emulates packet loss and delay on a Linux system. The server runs the Apache web server and hosts video chunk files. Chunks are stored in multiple directories based on their bitrates according to the MDP file[3].

In the following subsections, the default parameters of our network are as follows. The bandwidth between the client and server is set to 10 Mbps, packet loss is set to 1%, and RTT is set to 50 ms. Note that the size of each chunk depends on the throughput obtained when downloading the preceding chunk. If the measured throughput is high, then the client requests a chunk with higher bitrate which also has a larger size. This form of rate adaption is at the core of *adaptive* video streaming such as in DASH.

The throughput estimation is obtained by computing the average throughput of each chunk during transmission of the entire video file from the server to client. The video file is divided into 40 chunks each of which contains 15 seconds of the actual video. The throughput estimate is then obtained by calculating the average of 40 throughputs computed for the chunks.

III. Measurement Results and Discussion

A. Effect of Packet Loss

Fig. 5 shows the effect of loss probability on the aggregate throughput when using parallel connections. As discussed earlier, the use of multiple connections alleviates the negative effect of packet loss on TCP throughput. It is observed that as the number of parallel connections increases, the aggregate throughput converges to a value close to the link capacity regardless of the actual loss probability. In particular, when the loss probability is high (as in wireless networks), the use of parallel connections is more effective.

[3]The video file and dataset is obtained from http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/

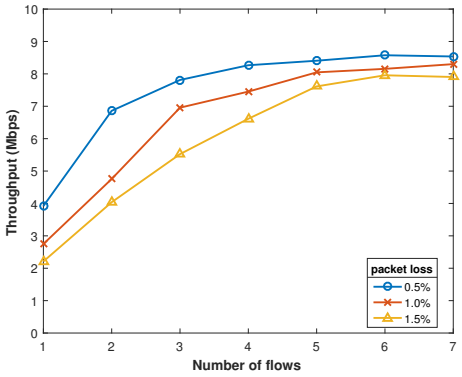


Fig. 5: Effect of packet loss on aggregate throughput.

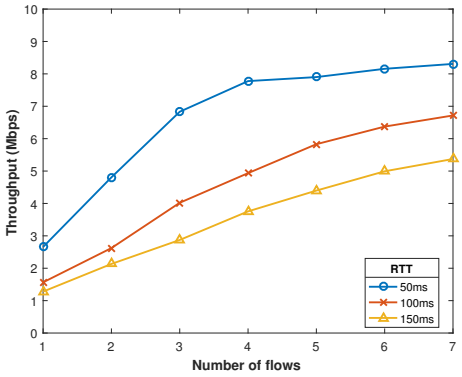


Fig. 6: Effect of RTT on aggregate throughput.

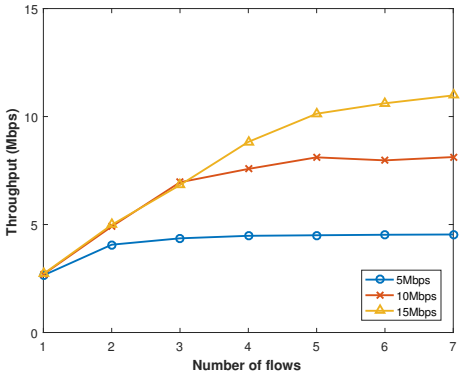


Fig. 7: Effect of link bandwidth on aggregate throughput.

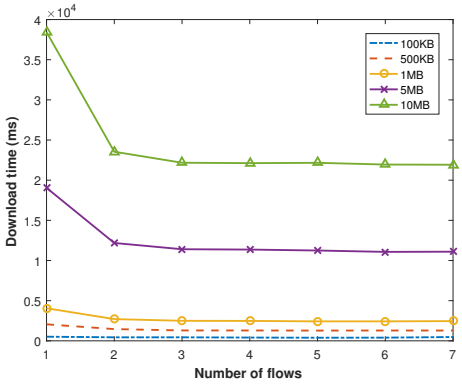


Fig. 8: Effect of chunk size on download time.

B. *Effect of Round Trip Time*

It is well-known that as RTT increases the TCP throughput decreases. Fig. 6 shows the effect of RTT on the aggregate throughput when using parallel connection. Again, as the number of connections increases so does the aggregate throughput though the amount of improvement somewhat depends on the value of RTT. In particular, in scenarios when the RTT is large (*e.g.*, in high-latency networks) more connections are required to achieve high utilization of the link. The reason is that TCP performs so poorly in such scenarios that there is a lot of room for improvement by adding more connections.

C. *Effect of Bandwidth*

The effect of the available bandwidth between the client and server on the aggregate throughput is depicted in Fig. 7. It is observed that using parallel connections helps utilize most of the available bandwidth as opposed to a single connection, which is oblivious to the link bandwidth in this scenario. It is also observed that as the available bandwidth increases, the number of parallel connections required to utilize the bandwidth increases as well. As the link bandwidth increases, TCP throughput becomes bounded by the loss probability and RTT. Thus, a single connection is not able to achieve higher throughput even if the link bandwidth increases beyond a limit. In this scenario, parallel connections are necessary to utilize the full bandwidth.

D. *Effect of Chunk Size*

Fig. 8 depicts the average time to download a chunk for different chunk sizes. As expected, increasing the number of parallel connections results in lower download time. However, as can be seen, only a few connections (*e.g.*, two) are enough to achieve most of the benefits of parallel connections. An important observation is that as the chunk size increases using parallel connections becomes more effective. Note that when streaming video at high quality, each chunk has a larger size. Thus, using parallel connections is particularly useful when streaming high quality video. A lower download time means less buffering delay for video playback, which directly translates to a better quality of experience.

E. *Transmit Queue Occupancy*

The transmit queue of the router plays a critical role in determining the utilization of the bottleneck link. A higher occupancy for the queue indicates a higher link utilization in general. To show the effect of multiple flows on the transmit queue occupancy, we run a simulation experiment with transmit queue size set to 100 packets.

Transmit queue traces over a period of 20 seconds are presented in Fig. 9. It can be seen that, while the queue occupancy fluctuates over time, having multiple flows generally results in higher occupancy rates. In Table I, we show the average number of packets in the queue (*i.e.*, the average

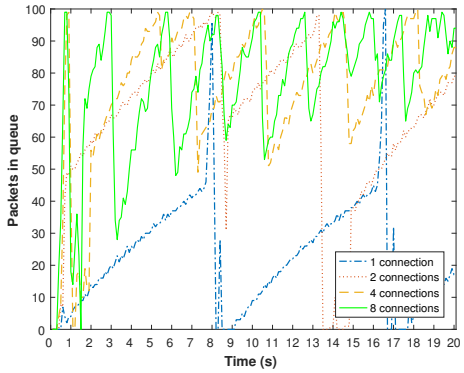


Fig. 9: Transmit queue occupancy with different number of flows. The size of the queue is set to 100 packets.

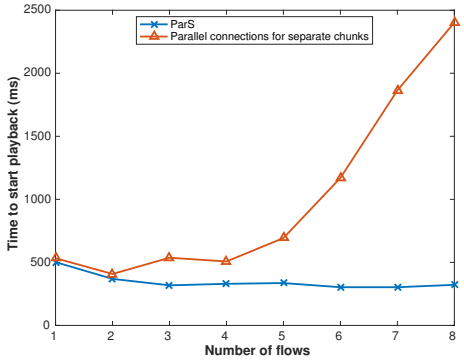


Fig. 10: Comparison of playback start time between parallel connections for the same chunk (ParS) and parallel connections for separate chunks (Parallel). Bandwidth=10 Mbps, RTT=50 ms, Loss=1 %.

queue occupancy) for different number of flows along with the achieved aggregate throughput. As expected, increasing the number of parallel flows results in increased average queue occupancy. Consequently, as the queue occupancy increases so does the aggregate throughput.

TABLE I: Average queue occupancy from simulations.

No. flows	Tput (Mbps)	Avg. packets in queue
1	8.839	22.62
2	9.623	64.49
4	9.785	74.64
8	9.837	76.45

F. Time To Start Playback

Time to start playback of a video is one of the main QoE factors in video streaming. It is defined as the time spent since a client requests a video to play, till the playback of the video by the player application begins on the client side. Having parallel connections in HTTP video streaming as discussed earlier, can be implemented in such a way that each parallel connection requests and downloads a separate chunk. We refer to this approach as *Parallel*. We compare Parallel with our method, ParS, which uses multiple connections to download different portions of the same chunk.

Fig. 10 depicts the measured time to start playback for these two methods. It is observed that as the number of parallel connections increases, time to start playback for Parallel increases as well. The longer time to start playback has a negative impact on QoE specially in low bandwidth networks (*e.g.*, wireless networks). Interestingly, with Pars, the time to start playback remains fairly constant and less than 500 milliseconds. Moreover, increasing the number of connections actually results in shorter time to start playback for ParS.

IV. Conclusion

In this work, we introduced ParS, a video streaming system that uses parallel TCP connections in conjunction with HTTP range requests to speedup downloading video chunks from the server. We implemented a prototype of the system on a testbed and conducted various measurement experiments to study its performance. In future, we plan to extend our implementation to mobile devices and conduct live measurements on WiFi and LTE networks.

References

[1] "Cisco visual networking index: Forecast and methodology." [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-gener

[2] "Adobe dynamic streaming." [Online]. Available: http://www.adobe.com/products/hds-dynamic-streaming.html

[3] "Apple HLS." [Online]. Available: https://developer.apple.com/streaming/

[4] "Microsoft smooth streaming." [Online]. Available: https://www.microsoft.com/silverlight/smoothstreaming/

[5] "Dynamic adaptive streaming over HTTP (DASH) part 1: Media presentation description and segment formats," online, 2014, iSO/IEC 23009-1:2014.

[6] G. Tian and Y. Liu, "On adaptive HTTP streaming to mobile devices," in *IEEE PV*, 2013.

[7] R. Kuschnig, I. Kofler, and H. Hellwagner, "Improving internet video streaming performance by parallel tcp-based request-response streams," in *IEEE CNCC*, 2010.

[8] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving tcp/ip performance over wireless networks," in *ACM MobiCom*, 1995.

[9] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "Gridftp: Protocol extensions to ftp for the grid," *Global Grid ForumGFD-RP*, vol. 20, 2003.

[10] C. Liu, I. Bouazizi, and M. Gabbouj, "Parallel adaptive HTTP media streaming," in *IEEE ICCCN*, 2011.

[11] M. Seufert *et al.*, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, 2015.

[12] "Internet download manager." [Online]. Available: http://www.internetdownloadmanager.com

[13] "Download accelerator plus." [Online]. Available: http://www.speedbit.com

[14] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner, "Adaptive streaming over content centric networks in mobile networks using multiple links," in *IEEE ICC*, 2013.

[15] S. Smanchat, K. Sangkul, and J. Y. Tham, "Enabling parallel streaming of multiple video sections by segment scheduling," in *ACM MoMM*, 2015.

[16] A. K. Chaurasia and A. K. Jagannatham, "dynamic parallel tcp for scalable video streaming over mimo wireless networks," in *IFIP WMNC*, 2013.