MPTCP+: Enhancing Adaptive HTTP Video Streaming over Multipath

Jia Zhao[1], Jiangchuan Liu[1], Cong Zhang[2], Yong Cui[3], Yong Jiang[4] and Wei Gong[2]
[1]School of Computing Science, Simon Fraser University, Canada
[2]School of Computer Science and Technology, University of Science and Technology of China, China
[3]Department of Computer Science and Technology, Tsinghua University, China
[4]Graduate school at Shenzhen, Tsinghua University, China

***Abstract*—This paper presents a systematic study on adaptive streaming over MPTCP. We start from realworld experiments with Dynamic Adaptive Streaming over HTTP (DASH) and analysis on its performance over MPTCP. We show that DASH can greatly benefit from the improved aggregated throughput by MPTCP; yet the inter-path throughput difference and the intra-path throughput fluctuation have noticeable (negative) impact, too. Without a proper design of path selection and adaptation in MPTCP, they can easily confuse the adaptation logic of DASH, resulting in low bitrates or frequent rebuffering even if high-bandwidth paths are available. We present MPTCP+, an extended multipath TCP solution to offer high quality and smooth playback for adaptive HTTP streaming. MPTCP+ incorporates a path use decision algorithm that smartly disables/enables a path to minimize the inter-path difference, and a novel congestion control algorithm that smooths congestion window evolution with multiple paths. We have implemented MPTCP+ in the MPTCP Linux kernel, with minimum change on the server-side MPTCP module only. It is fully compatible with the existing MPTCP clients and requires no change on the upper-layer protocols, too. Our experiments suggest that MPTCP+ increases the quality of experience (QoE) of DASH by up to 50%.**

I. Introduction

HTTP-based adaptive video streaming has become the most popular solution for online video applications, and it is still having a rapid growth in the Internet traffic. For example, in the peak evening hours, HTTP video streaming traffic (from the services such as Netflix and YouTube) accounts for more than 60% of all downstream traffic worldwide according to the global Internet report [1, 2]. In fact, adaptive bitrate (ABR) is efficient for video content delivery under varying network conditions. First, online video viewers use heterogeneous networks with different capacity to fetch video content, and it is difficult to satisfy diverse quality demands with the same bitrate. Second, fixed bitrate could be too aggressive when network conditions drop to a poor level (due to congestion or link transport error), thus degrading video quality of experience (QoE). In contrast, ABR can detect network conditions and intelligently switch between multiple bitrates. The standardized protocol Dynamic Adaptive Streaming over HTTP (DASH) [3], also known as MPEG-DASH, has been widely adopted by online video services. DASH does not need specialized media servers, and media content can be delivered over widely deployed HTTP-based Content Delivery Networks (CDN). In addition, since DASH's data is encapsulated with regular TCP and HTTP headers, it can traverse firewalls and NAT well in the Internet [4].

End-to-end throughput provided by transport layer is fundamentally critical to DASH video quality. While there have been many application-layer approaches that improve DASH ABR logic [5–7], recent work has started learning transport protocol's throughput pattern during DASH video delivery [8]. High and stable throughput of a transport protocol is always desirable for DASH videos. In fact, transport throughput decides the quality of video streaming (e.g., 1Mbps for 480p H.264 videos, and 5Mbps for 1080p H.264 videos), and stable throughput can also avoid frequent bitrate switches. Nevertheless, the conventional TCP is hard to serve these goals. It has been discussed in [9] that an adaptive video stream over TCP can have abrupt throughput drop when coexisting with other competing flows and this will cause low average throughput. Also, many congestion control designs for video applications seek to improve the highly fluctuating sending rate of TCP [10]. Given the importance and high-quality demand of adaptive streaming application in the current Internet, it is necessary to redesign the transport protocol that can offer high and stable throughput for adaptive streaming, without losing compatibility and fairness to other applications.

Multipath TCP (MPTCP) is a transport protocol that enables an end-to-end TCP-based connection to use multiple paths for data transmission, and it has been published as RFCs [11–13]. MPTCP matches the multihoming network scenarios, e.g., ubiquitous mobile devices equipped with both WiFi and cellular interfaces. Compared with regular TCP, MPTCP brings the advantages: it can obtain aggregate bandwidth from concurrent usage of multiple paths, thus increasing throughput [14, 15], and it can also improve the window fluctuation of TCP [16]. Hence, MPTCP has great potential to serve DASH video streams. However, the practical performance of MPTCP for adaptive streaming application remains largely unclear to the research community.

In this paper, we first analyze the performance of DASH over MPTCP on a realworld testbed, and we find that although MPTCP can significantly increase aggregate throughput there are also challenges. First, large inter-path bandwidth difference confuses the ABR logic of DASH. The low-throughput path

978-1-7281-6887-6/20/$31.00 © 2020 IEEE

978-1-7281-6887-6/20/$31.00 ©2020 IEEE

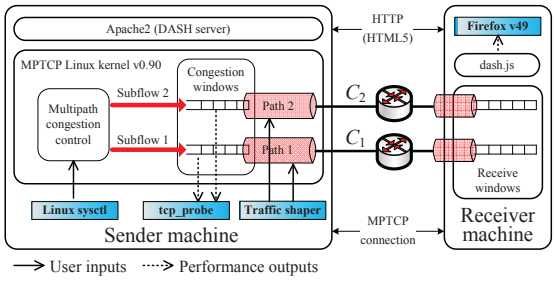


Fig. 1. Testbed setup

increases the download duration of a video segment and feeds back to the ABR logic to reduce bitrates. This will result in low bitrates or rebuffering even if there is a high-bandwidth path. Second, smooth playback suffers from the throughput fluctuation of MPTCP. The violent fluctuation of subflow congestion window (cwnd) can be added and exerted on the total cwnd. This may lead to frequent bitrate switches in DASH's ABR logic. While previous work has studied the fixed bitrate streaming over MPTCP [17] and the preference-aware MPTCP for DASH [18], the above problems have never been investigated in the literature but could degrade DASH video streaming quality.

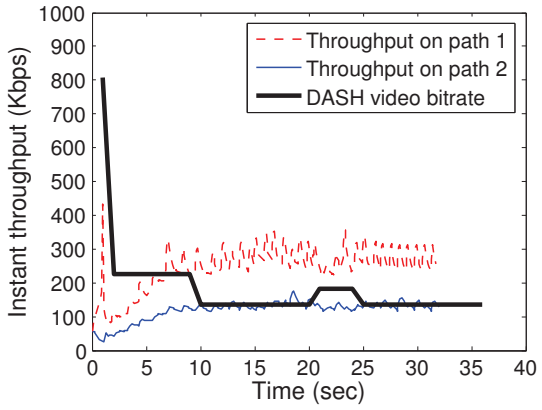
Motivated by these problems, we design, implement and evaluate MPTCP+, an extended multipath TCP solution for adaptive streaming. The goal of MPTCP+ is to serve high quality and smooth playback of adaptive streaming. MPTCP+ incorporates the path use decision (PUD) and the multipath congestion control (MCC). PUD enables a path if its estimated throughput is close enough to the highest throughput among paths and disables it otherwise. PUD uses the sampled socket information of each MPTCP subflow to calculate the average throughput on each path. MCC introduces a smoothing factor in the coupling of all subflow cwnds and alleviates the sending rate fluctuation. MCC uses a Round Trip Time (RTT) based condition to trigger the smoothing for both increase and decrease of each subflow cwnd. We implement MPTCP+ in the MPTCP Linux kernel. MPTCP+ has minimum change on the server-side MPTCP module only. It is fully compatible with the existing MPTCP clients and requires no change on the upper-layer protocols, too. We discuss the parameter selection of MPTCP+ and evaluate its performance in realworld experiments. Our results show that MPTCP+ outperforms MPTCP in both video segment download time and throughput, and it can increase the QoE of DASH streaming by up to 50% compared to MPTCP in different network conditions.

The remainder of this paper is organized as follows. Section II analyzes the performance of DASH over MPTCP. In section III, we present the design of MPTCP+. Section IV discusses the parameter selection and evaluates MPTCP+ in the experiments. Section V concludes the paper.

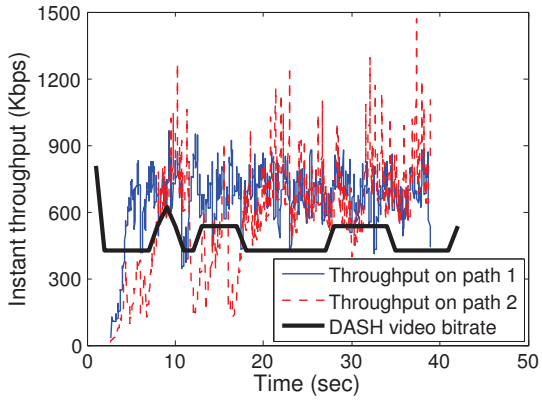
II. DASH over MPTCP: Challenges

A. Inter-path Throughput Difference

We first examine how inter-path throughput difference impacts the video quality. As shown in Fig. 1 (see the detailed testbed setup in section IV), there are two available paths for MPTCP. We limit the download speeds, so that the two paths can have different configurations, i.e., two paths with large throughput difference or with small throughput difference. For each configuration, we record the instant throughput on each path and the DASH video bitrates.



(a) $\overline{x}_2/\overline{x}_1 \approx 0.486$



(b) $\overline{x}_1/\overline{x}_2 \approx 0.869$

Fig. 2. DASH over MPTCP: $\overline{x}_1$ and $\overline{x}_2$ are average throughputs of the two paths; Large throughput difference cause low bitrates of DASH.

Fig. 2 shows the results. We calculate the relative throughput difference (i.e. the ratio of two path throughputs). For the two cases in Fig. 2, DASH starts from the same bitrate 808.057Kbps, which is a moderate level among the bitrate representations. After the first bitrate, DASH will request the next bitrates by following the decision of its ABR logic. The ABR logic is unaware of multipath at the transport layer. As shown in Fig. 2(a), the throughput difference between the two paths is $\overline{x}_1/\overline{x}_2 \approx 0.486$, and DASH sensitively reduces the bitrates to a low level. We can see that the bitrates are constrained by the low-throughput path. When the throughput difference becomes smaller ($\overline{x}_1/\overline{x}_2 \approx 0.869$) as shown in Fig. 2(b), there is no obviously low-throughput path that can affect the DASH video bitrates. Our results indicate the performance degradation of DASH streaming in large inter-path throughput difference. This motivates us to find an efficient approach to timely disable the low-throughput path when there is large throughput difference among the used paths. Since the

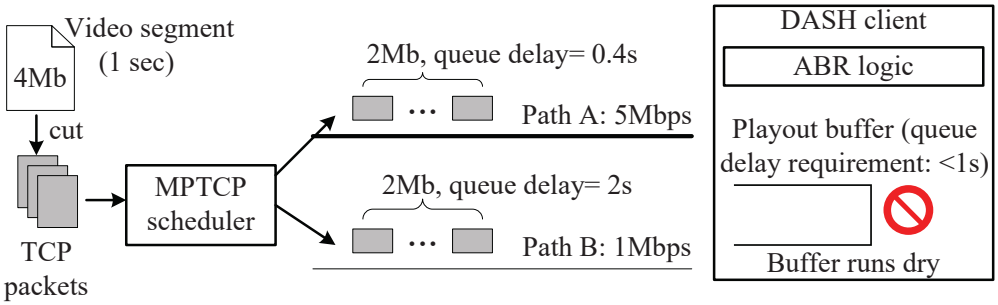
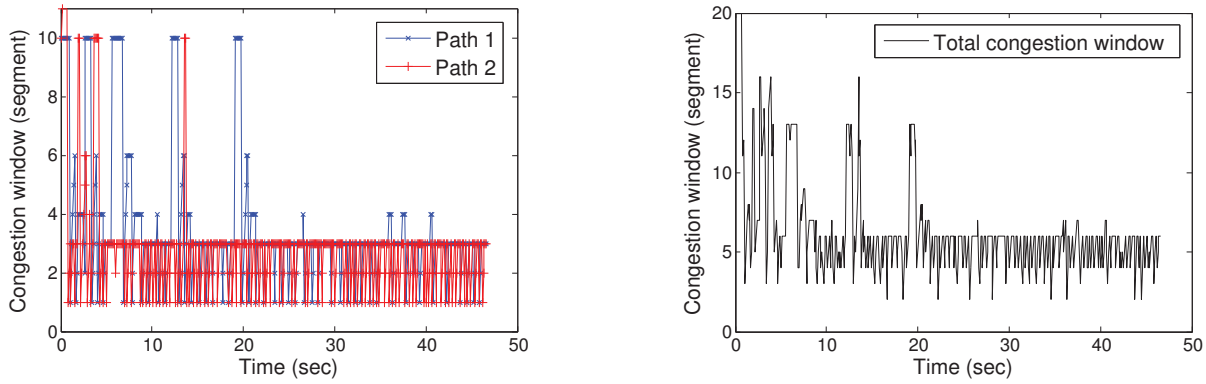


Fig. 3. Rebuffer event and low bitrate resulting from large inter-path throughput difference.



(a) cwnds on two paths

(b) Total cwnd

Fig. 4. Violent window fluctuation of MPTCP for the first 47 seconds DASH video transfer: each path has download speed limit 500Kbps.

experimental results apply to adaptive streaming rather than general data transmission, the modification of MPTCP path usage should be at DASH server side.

We first analyze the inefficiency of low throughput path (i.e., why the low throughput path leads to the decrease of DASH bitrates) in the multipath video transfer. Video player uses the playout buffer to maintain constantly non-interrupted video playback. The buffer size is usually limited to a few seconds. Fig. 3 shows an example of the inefficiency of using multiple paths with large throughput difference: the video segment (4Mbps bitrate) is first cut and packed into multiple TCP packets, and then each packet is assigned to a path (with capacity of 5Mbps or 1Mbps) by the MPTCP scheduler. Clearly, the packets sent over the low-throughput path will increase the download time of their belonging video segment. Then the video segment delay will make the DASH's ABR logic (*ThroughputRule* and *AbandonRequestsRule* [20]) select low bitrates for the next segment. This is because the ABR algorithms [20] use per-segment throughput or delay to decide bitrates. Even worse, if the delay exceeds the requirement of the playout buffer, the rebuffer event may happen.

B. Intra-path Window Fluctuation

In addition to high bitrate, smooth playback is also important for the quality of adaptive streaming. Multiple paths with similar throughputs can avoid low video bitrates, but bitrate oscillations may happen due to highly fluctuating throughput. As shown in Fig. 2(b), the instant throughputs of the two paths experience large variation, and the DASH video suffers from frequent bitrate switches. Although DASH can benefit from high aggregate throughput of MPTCP, its playback may suffer from violent throughput fluctuation brought by the throughput aggregation. Each subflow maintains its own cwnd and runs the coupled congestion control algorithm on its belonging path. The coupled multipath congestion control introduces the total cwnd, which is the sum of the subflow cwnds. The subflow cwnd fluctuation can be added and exerted on the total cwnd, resulting in the fluctuation of the aggregate throughput. If the total cwnd has a highly fluctuating pattern, it will trigger DASH's ABR logic to switch bitrate frequently, and it will lead to bitrate oscillations and instable quality of DASH streaming. As shown in Fig. 4, there are highly fluctuating subflow cwnds during the DASH video playback, and this causes violent fluctuation of the total cwnd and severe bitrate oscillations.

III. Design of MPTCP+

A. Path Use Decision

PUD can stop using low-throughput paths once it detects unacceptable inter-path throughput difference. First, it estimates the throughput of each path. Then it calculates the inter-path throughput difference and disable the low-throughput paths if necessary.

PUD module first smooths the sampled throughput data. By using the smoothing, we can reduce short-term fluctuations.

Algorithm 1: Path Use Decision

Input: TCP socket of each subflow r: sub_sk
Output: Set of allowed paths: A

Initialization:
$max\_mean\_SendRate \leftarrow 0$ $num\_samples \leftarrow n$
for *each subflow* r **do**
  $rtt[r] \leftarrow 0$; $SendCwnd[r] \leftarrow 1MSS$;
  $SendRate[r] \leftarrow 0$; $sum\_w\_SendRates[r] \leftarrow 0$
  $mean\_SendRate[r] \leftarrow 0$; $sum\_last2[r] \leftarrow 0$
Decision on allowed paths:
for *each subflow* r **do**
  for *each sample* i *of subflow* r **do**
    $rtt[r] = read\_rtt(sub\_sk)$;
    $SendCwnd[r] = read\_SendCwnd(sub\_sk)$;
    $SendRate[r] = MSS * SendCwnd[r]/rtt[r]$;
    /*refer to DASH's ThroughputRule to average the throughputs of the last three segments*/
    if $i = 1$ **then**
      /*smooth send rate*/
      $S\_SendRate[i] = SendRate[r]$;
    if $i = 2$ **then**
      $S\_SendRate[i] = (sum\_last2[r] + SendRate[r])/2$;
    if $2 < i \leq n$ **then**
      $S\_SendRate[i] = (sum\_last2[r] + S\_SendRate[i])/3$;
    update $sum\_last2[r]$;
    /*sum of weighted send rates*/
    $sum\_w\_SendRates[r] = +i * S\_SendRate[i]$;
    /*update the maximum average throughput of multiple subflows*/
    if $\frac{2*sum\_w\_SendRates[r]}{i(i+1)} > max\_mean\_SendRate$ **then**
      $max\_mean\_SendRate = \frac{2*sum\_w\_SendRates[r]}{i(i+1)}$;
  $mean\_SendRate[r] = \frac{2*sum\_w\_SendRates[r]}{n(n+1)}$;
  if $mean\_SendRate[r] > \rho * max\_mean\_SendRate$ **then**
    add path r to the allowed path set A;
  else
    $SendCwnd[r] = 0$;
return A

We adopt the smoothing algorithm Moving Average (MA), which calculates the average of a sample subset with fixed size and shifts the calculation across the time series samples. We use the average of the last three sampled throughput values.

Let n be the number of samples. PUD uses the weighted average of the MA throughput values. PUD gives the MA of the nth sample the largest weight n, and it gives the ith sample $(1 \leq i < n)$ a weight i. This is because we want to weaken the influence of the samples taken during the initial playback and emphasize the importance of the latest samples. Let $\overline{x}_r$ be the weighted throughput average of each subflow r, and it is expressed as $\overline{x}_r = \frac{\sum_{i=1}^{n} MA_r^{(i)} \cdot i}{n(n+1)/2}$.

PUD calculates the average throughput $\overline{x}_r$ for each subflow r and obtains the maximum $\max_r \overline{x}_r$. Then PUD uses a parameter ρ to set the threshold of throughput. If $\overline{x}_k \leq \rho \max_r \overline{x}_r$, the congestion window of subflow k will be reduced to zero. If $\rho \max_r \overline{x}_r < \overline{x}_k \leq \max_r \overline{x}_r$, subflow k will be added to the allowed path set. In section VI, we will discuss the selection of parameters n and ρ. The pesudo code for implementation is summarized in Algorithm 1. More implementation details are introduced in section V.

B. Multipath Congestion Control

MPTCP+ does not directly use the existing multipath congestion control algorithms such as LIA [19], OLIA [15] and Balia [16]. This is because our design goal is to provide high and stable throughput. In order to achieve stable throughput, we design the congestion control that can smooth the total congestion window and deal with the highly fluctuating window performance.

We use the network model which is similar to [15]. MPTCP+ should ensure TCP-friendliness and have less fluctuating window evolution. These design goals lead us to propose the following fluid model:

$$\frac{dx_r}{dt} = \theta x_r^2 \Big(\frac{1}{RTT_r^2 (\sum_{k \in s} x_k)^2} - \frac{1}{2} \lambda_r \Big) \tag{1}$$

where θ is the smoothing factor and it is expressed as follows:

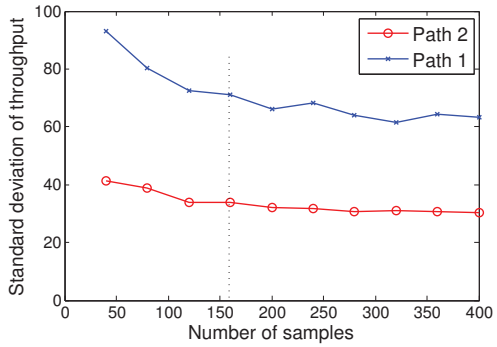
$$\theta = \begin{cases} \frac{\min_{k \in s} w_k}{\sum_{k \in s} w_k} & \text{if } RTT_r > 1.5\Gamma_r \\ 1 & \text{else} \end{cases} \tag{2}$$

Γ_r is the weighted mean of last ten RTTs. We now illustrate how the fluid model serves our design goals. According to the theoretical analysis in [19], RFC 6356 [12] and [15], our model satisfies the TCP friendliness requirement.

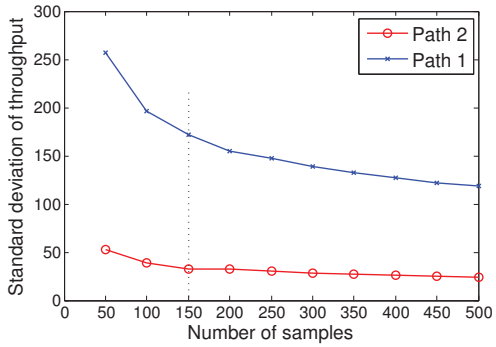
Next we illustrate how the factor θ smooths the window fluctuation. Peng *et al.* [16] study the properties of multipath congestion control algorithms and generalize the existing algorithms including LIA [19], OLIA [15] and Balia [16] with the following fluid model:

$$\frac{dx_r}{dt} = \varphi_r(\mathbf{x}_s)(\phi_r(\mathbf{x}_s) - \frac{1}{2}\lambda_r) \tag{3}$$

where $\phi_r(\mathbf{x}_s)$ determines the equilibrium properties, and $\varphi_r(\mathbf{x}_s)$ determines the dynamic properties including window fluctuation. LIA [19], OLIA [15] and Balia [16] all have $\varphi_r(\mathbf{x}_s) = x_r^2$, and they are all loss-based congestion control algorithms. In [16], the authors use $\varphi_r(\mathbf{x}_s)$ as a measure of window fluctuation, and it has been discussed that smaller $\varphi_r(\mathbf{x}_s)$ means smaller window fluctuation. Our model in Equation (1) can also be generalized by Equation (3), and we have $\varphi_r(\mathbf{x}_s) = \theta x_r^2$. Because $\theta \leq 1$ (Equation (2)), our algorithm can have smaller window fluctuation.

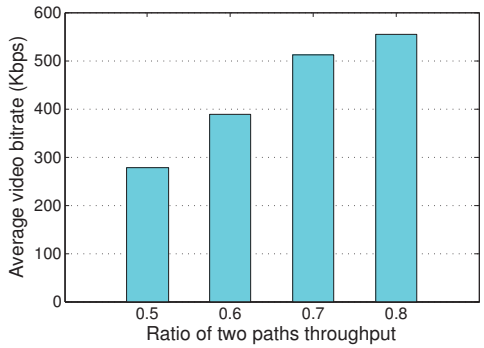


(a) $\overline{x}_1 = 867.5$Kbps, $\overline{x}_2 = 461.5$Kbps

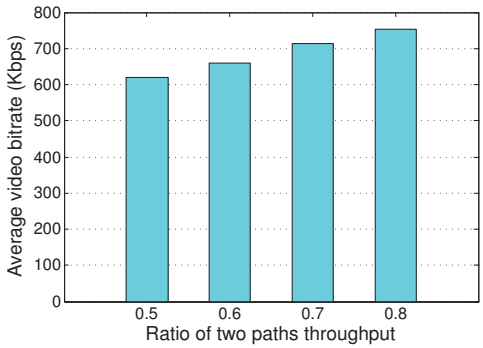


(b) $\overline{x}_1 = 3141.7$Kbps, $\overline{x}_2 = 647.4$Kbps

Fig. 5. Standard deviation of throughput change with number of samples. Path 1 has average throughput $\overline{x}_1$, and path 2 has average throughput $\overline{x}_2$.



(a) $C_2 = 500$Kbps



(b) $C_2 = 1000$Kbps

Fig. 6. DASH video quality change with inter-path download speed difference (C_1/C_2).

IV. Performance Evaluation

Our testbed is an end-to-end DASH video streaming system. As shown in Fig. 1, the system includes the server and client machines, both equipped with double network interfaces. We install Ubuntu 14.04 and the MPTCP Linux kernel of version 0.90 on each machine. We use the default LowRTT scheduler in MPTCP. We use the command-line based network traffic shaping tool wondershaper [21] to change download and upload speeds. We use the Linux kernel module tcpprobe to record the state of MPTCP subflows during video transmission and capture the time-varying variables such as instantaneous throughput and cwnd on each path. At the server side, we use Apache2 [22] as our HTTP server. We use the open dataset of DASH videos [23, 24]. The dataset uses the source video quality of 1080p YUV and H.264/MPEG-4 AVC codec. The source video has a length of 596 seconds and is chunked into 596 video segments. Each video segment has a length of one second and is encoded with 20 bitrate levels ranging from 46.980Kbps to 4726.737Kbps [24]. At the client side, we use dash.js [25] as the DASH player. The video playback calls dash.js and uses the Media Source Extensions (MSE) supported by Firefox browser of version 49 [26].

A. Parameter Selection

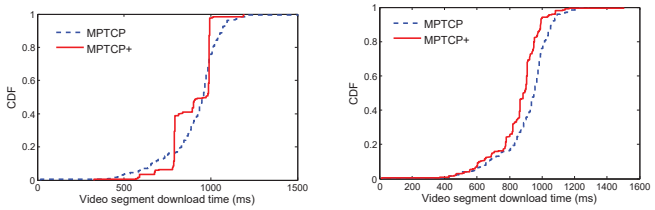
The PUD algorithm in subsection III.A needs to set two parameters: the number of samples n and the threshold ρ.

Let x_i be the throughput of sample i and $\overline{x}$ be the average throughput. Fig. 5 shows that the standard deviation changes with the number of samples n in different throughput patterns. We use MPTCP to transmit DASH video segments. Path 1 has high throughput and Path 2 has low throughput. Fig. 5 shows that the high throughput path has large standard deviation. Both the paths have relatively stable deviation when the number of samples becomes large. We see that 150 samples are enough for a path to arrive at its stable state.

Next we select the parameter ρ. Fig. 6 shows the DASH video quality change with inter-path throughput difference. As we make the ratio C_1/C_2 change from 0.5 to 0.8, the video quality (measured by the average of ever used bitrates) decreases with the inter-path throughput difference. We see that the negative impact of inter-path throughput difference is still there at $C_1/C_2 = 0.5$ and $C_1/C_2 = 0.6$, especially when we use relatively slow download speed (Fig. 6(a)). When $C_1/C_2 > 0.6$, we achieve better video quality. Considering the overall performance, we set the threshold $\rho = 0.6$.

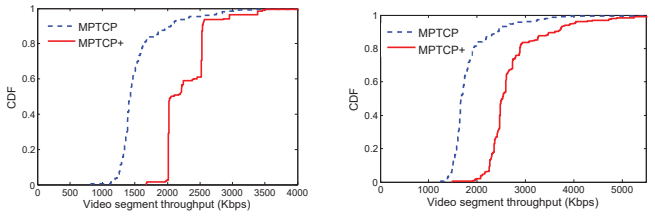
B. QoE of DASH Video Playback

We examine the video segment download delay and throughput of DASH over MPTCP+ and MPTCP, respectively. The traces of video segment size and download delay are recorded by the network inspector of Firefox browser of version 49 [26]. We control the path bandwidth and change



(a) C_1=150, C_2=1500 (b) C_1=750, C_2=1500

Fig. 7. QoE measured by DASH video segment download delays.



(a) C_1=150, C_2=1500 (b) C_1=750, C_2=1500

Fig. 8. QoE measured by DASH video segment download speed.

the ratio of two paths' bandwidth values. Fig. 7 compares video segment delays (using Cumulative Distribution Function (CDF)) for MPTCP+ and MPTCP. The results show that MPTCP+ outperforms MPTCP from the perspective of segment download delay. Video segments on low-throughput paths increase their download delays. Our results indicate that MPTCP+ can disable the low-throughput path and thus obtain better download performance. Fig. 8 compares segment download speeds for MPTCP+ and MPTCP during DASH video playback. MPTCP+ achieves higher segment download speed than MPTCP, especially in the case of 1500Kbps average bandwidth and large inter-path throughput difference (as shown in Fig. 8(a)). Note that we use the software tool wondershaper to set the traffic limits C_1 and C_2, which can be described as the average bandwidth. The instantaneous throughput and single video segment download speed can be much higher than the average bandwidth.

V. Conclusion

In this paper, we have studied the performance of MPTCP for DASH video streaming and proposed a variant of MPTCP to address the problems of inter-path throughput difference and congestion window fluctuation. The proposed solution MPTCP+ can use its path use decision module to disable very low throughput paths and help DASH's ABR logic to increase bitrate. The multipath congestion control module of MPTCP+ can improve congestion window fluctuation that may lead to DASH bitrate oscillations. The realworld experiment results have shown that MPTCP+ can improve video segment download throughput and increase the QoE of DASH video playback. The future work will be concerned with efficient and fair transport layer solution for bottleneck-shared competing DASH streams.

Acknowledgment

We would like to thank the anonymous reviewers for valuable and insightful comments. This research is supported by Canada NSERC Discovery Grant. C. Zhang's work was supported by the National Natural Science Foundation of China (No. 61902369). Y. Cui's work was supported by the National Natural Science Foundation of China (No. 61872211). Y. Jiang's work was supported by Shenzhen Nanshan District Ling-Hang Team Grant under No. LHTD20170005.

References

[1] https://www.sandvine.com/phenomena
[2] http://digg.com/2018/streaming-video-worldwide
[3] ISO/IEC 23009-1 Second edition, Dynamic Adaptive Streaming over HTTP (DASH), 2014.
[4] B. Li, Z. Wang, J. Liu, and W. Zhu, "Two Decades of Internet Video Streaming: A Retrospective View," *ACM Trans. Multimedia Computing, Communications and Applications*, vol. 9, no. 1s, pp. 33-52, 2013.
[5] G. Tian and Y. Li, "Towards Agile and Smooth Video Adaption in Dynamic HTTP Streaming," in *Proc. ACM CoNEXT*, 2012.
[6] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," in *Proc. ACM SIGCOMM*, 2014.
[7] Y. Zhang and Y. Liu, "Buffer-Based Reinforcement Learning for Adaptive Streaming," in *Proc. IEEE ICDCS*, 2017.
[8] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction," in *Proc. ACM SIGCOMM*, 2016.
[9] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard," in *Proc. ACM IMC*, 2012.
[10] L. Cai, X. Shen, J. Pan, and J. W. Mark, "Performance Analysis of TCP-friendly AIMD Algorithms for Multimedia Applications," *IEEE Trans. Multimedia*, vol. 7, no. 2, pp. 339-355, 2005.
[11] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," IETF RFC 6182, 2011.
[12] C. Raiciu, M. Handly, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," IETF RFC 6356, 2011.
[13] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF RFC 6824, 2013.
[14] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proc. ACM SIGCOMM*, 2011.
[15] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J. L. Boudec, "MPTCP is Not Pareto-optimal: Performance Issues and A Possible Solution," in *Proc. ACM CoNEXT*, 2012.
[16] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, Design, and Implementation," *IEEE/ACM Trans. Networking*, vol. 24, no. 1, pp. 596-609, 2016.
[17] J. Wu, B. Cheng, and M. Wang, "Energy Minimization for Quality-Constrained Video with Multipath TCP over Heterogeneous Wireless Networks," in *Proc. IEEE ICDCS*, 2016.
[18] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "MP-DASH: Adaptive Video Streaming over Preference-Aware Multipath," in *Proc. ACM CoNEXT*, 2016.
[19] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *Proc. Usenix NSDI*, 2011.
[20] https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic.
[21] https://github.com/magnific0/wondershaper.
[22] https://httpd.apache.org/.
[23] S. Lederer, C. Mller, and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *Proc. ACM MMSys*, 2012.
[24] http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/ BigBuckBunny/1sec/.
[25] https://github.com/Dash-Industry-Forum/dash.js/.
[26] https://www.mozilla.org/en-US/firefox/49.0/releasenotes/.