On the FullMesh Path Selection of Multipath TCP Video Streaming

Yosuke Komatsu*, Dirceu Cavendish**, Daiki Nobayashi**, Takeshi Ikenaga**
*Graduate School of Engineering, **Faculty of Engineering
Kyushu Institute of Technology
Fukuoka, Japan
e-mail: komatsu.yosuke620@mail.kyutech.jp, {nova@ecs, ike@ecs, cavendish@net.ecs}.kyutech.ac.jp

***Abstract*—Video streaming makes most of Internet traffic nowadays, with most video applications transported over Hypertext Transfer Protocol/Transmission Control Protocol (HTTP/TCP). Being the predominant transport protocol, TCP stack performance in transporting video streams plays an important role, especially with regard to MultiPath Transport Control Protocol (MPTCP) and multiple client device interfaces currently available. One overlooked aspect of multipath transport is the management of all possible paths between sender and receiver endpoints. In this paper, we study the usage of all possible paths created by MPTCP vis a vis video streaming performance on wired networking environments. We show that a fullmesh path usage may result in degraded video streaming performance due to common bottlenecks between paths under a simple (default) path scheduler. We then propose a bottleneck aware path scheduler, and show its superior performance on various multipath scenarios. Our results cover both Bottleneck Bandwidth and Round-trip (BBR) propagation time TCP variant, as well as CUBIC variant in transporting video streams over wired networks. We use network performance level, as well as video quality level metrics to characterize quality of video streaming over TCP variants.**

Keywords*—Video streaming; TCP congestion control; Multipath TCP; TCP BBR.

I. Introduction

Video streaming nowadays accounts for the majority of Internet traffic. Regarding streaming applications, video stream quality is related to two factors: the amount of data discarded at the client end point due to excessive transport delay/jitter and data rendering stalls due to lack of timely playout data. Transport delays and data starvation depend heavily on how Transport Control Protocol (TCP) handles retransmissions upon packet losses during flow and congestion control. Moreover, in multipath transport scenarios, it is important to manage head-of-line blocking across various networking paths, potentially with diverse loss and delay characteristics. Head-of-line blocking occurs when data already delivered at the receiver has to wait for additional packets that are blocked at another path, potentially causing incomplete or late frames to be discarded at the receiver, as well as stream rendering stalls. In addition, the various paths used in transporting the data may interfere with each other at times. In this paper, we study interference of multiple transport paths in a full mesh configuration, where all available networking paths are used for video transport. As transport delays and data starvation depend heavily on how TCP handles retransmissions upon packet losses during flow and congestion control, we analyze two TCP variants currently widely deployed: CUBIC [1] and BBR [2].

The paper is organized as follows. Related work is included in Section II. Section III describes video streaming transport over TCP, with focus to BBR and CUBIC TCP variants. Section IV describes the default path scheduler used in Linux environments, and introduces our new bottleneck aware path scheduler. Section V introduces these variants. Section VI characterizes video streaming performance over wired paths via network emulation. We compare the application and network performance of BBR against CUBIC, using the default (estimated shortest transmission time), as well as bottleneck aware path schedulers. Section VII summarizes our studies and addresses future directions to this work.

II. Related Work

Since MultiPath Transport Control Protocol (MPTCP) has become available, several multipath transport studies have appeared in the literature, mostly focusing on throughput performance of data transfers over mobile networks (see [3] and related work). [8] introduces path selection techniques, such as stickness (staying on a same path for as long as possible), in order to reduce head of line blocking in Video Streaming applications. Although the study shows improvements on wireless cellular/WiFi topology scenarios, it did not address interference among available paths.

[9] studies the performance of Adaptive Video Streaming (Dynamic Adaptive Streaming over HTTP (DASH)) on top of MPTCP transport. The adaptive bit rate nature of DASH causes large bit encoding fluctuations when paths of different throughput characteristics are present, causing bad video experience. The authors advocate blocking low throughput paths in order to stabilize adaptive bit rate and, improving Quality of Experience. The authors of [4] evaluate throughput of multipath video streaming over Digital Subscriber Line (DSL) multipath scenarios, without providing video level performance measures. Although they also propose a cost optimized scheduler, the lack of video quality performance measures limits conclusions about the impact of such a scheduler on video quality. Along the same lines, Imaduddin et al. [5] provide a performance evaluation of Multipath TCP (MPTCP) using CUBIC and Vegas TCP variants, as well as minimum Round Trip Time (RTT), round-robin and coupled Balia schedulers. Finally, Xing et al. [6] propose a new MPTCP scheduler which they show via network experiments to lower the number of out-of-order packets. The scheduler estimates receiver arrival times, and sends redundant packets to cope with estimation errors. Video streaming is simulated

via iperf3, and no application layer performance measures are used.

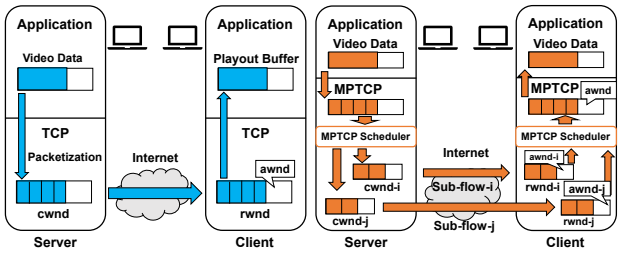
Regarding full mesh path selection, [10] studies the interference between paths sharing bottleneck resources, from a goodput performance perspective. They characterize inter-path positive and negative interference, and propose a "least interpath contention" path management strategy, where they limit the multipath transport to use only disjoint paths. They then use emulated single hop client/server testbed topology to evaluate full-mesh vs path disjoint goodput results vis a vis various couple and uncoupled congestion control schemes. In contrast, our work focuses on the intelligent path management taking into account common resources along fullmesh paths. Our path selection then strives to select paths in a timely manner so as not to cause interference even though paths may intersect.

III. Video Streaming Over MPTCP

A video application over Hypertext Transfer Protocol/Transmission Control Protocol (HTTP/TCP) starts at an HTTP server storing video content. At the transport layer, a TCP variant provides reliable transport of video data over IP packets between server and client end points (Figure 1 (a)). Upon HTTP video request, a TCP sender is instantiated to transmit packetized data to the client machine, connected to the application via a TCP socket. At the TCP transport layer, a congestion window is used at the sender to control the amount of data injected into the network. The size of the congestion window ($cwnd$) is adjusted dynamically, according to the level of congestion experienced on the network path, as well as space available for data storage ($awnd$) at the TCP client receiver buffer. Congestion window space at the sender is freed only when ACK packets acknowledging data packets are received. Lost packets are retransmitted by the TCP layer to ensure reliable data delivery. At the client, in addition to acknowledging arriving packets, the TCP receiver informs the TCP sender about its current receiver available space, so that $cwnd \leq awnd$ condition is enforced by the sender at all times to prevent receiver buffer overflow. At the client application layer, a video player extracts data from a playout buffer, which draws packets delivered by the TCP receiver from the receiver TCP socket buffer. The playout buffer hence serves to smooth out variable network throughput and delay. Multiple path transport brings communication reliability enhancements, as well as bandwidth increase. The challenge is video rendering degradation due to increase frame discards and buffer underflows originated from head of line blocking.

A. MPTCP

MPTCP is an Internet Engineering Task Force (IETF) extension of TCP transport layer protocol to support data transport over multiple concurrent TCP sessions when multiple interfaces are available [7]. The network multipath transmission of the transport session is hidden from the application layer by a legacy TCP socket exposed per application session. At the transport layer, however, MPTCP coordinates concurrent TCP



(a) TCP (b) MPTCP

Figure 1. Video Streaming over TCP/MPTCP.

sessions on various subflows (paths), each of which is itself unaware of the multipath nature of the application session. In order to accomplish multipath transport, a path scheduler connects the application socket with transport subflows, extracting packets from the application facing the MPTCP socket, selecting a subflow for transmission, and injecting packets into the selected subflow. The MPTCP transport architecture is depicted in Figure 1 (b).

The first and most used path scheduler, called default scheduler, selects the path with shortest RTT among paths with currently available congestion window space for new packets. Other path schedulers have appeared recently. These path schedulers can operate in two different modes: uncoupled, and coupled. In uncoupled mode, each subflow congestion window $cwnd$ is adjusted independently of other subflows. On the other hand, in coupled mode, the MPTCP scheduler couples the congestion control of the subflows, by adjusting the congestion window $cwnd_k$ of a subflow k according to the current state and parameters of all available subflows. Although many coupling mechanisms exist, we focus on the performance study of BBR [2] TCP variant over uncoupled schedulers in this work.

Regardless of the path scheduler used, IETF MPTCP protocol supports the advertisement of multiple IP interfaces available between two endpoints via specific TCP option signalling. IP interfaces may be of diverse nature (e.g., Wi-Fi, Long term evolution (LTE)). In addition, multipath transport requires an MPTCP stack at both endpoints for the establishment and usage of multiple paths. MPTCP signalling allows for a full mesh of connecting paths between two transport endpoints. That is, if there are N interfaces at the client and M interfaces at the server available, each N x M combination of interfaces constitutes a viable path.

IV. Path Schedulers

In this section, we use MPTCP default scheduler as a springboard to propose a novel bottleneck aware path scheduler.

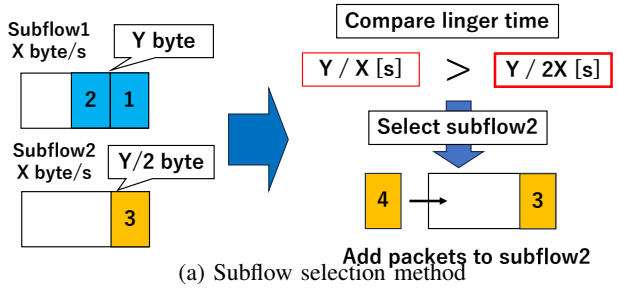
A. Default scheduler

An overview of the default scheduler algorithm is shown in Figure 2. The default scheduler (Linux kernel Version 6.1) selects the subflow that takes the least amount of time ($linger\_time$) to transmit all packets in the subflow buffer (Figure 3(a)). This time is calculated as:

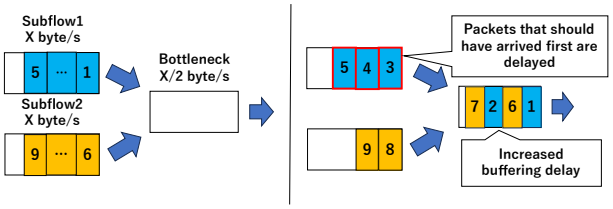
$$linger\_time = \frac{wmem}{pace} \quad (1)$$

Data: Set of subflows S
Result: Selected subflow S_{best}
Initialization:
  Set $best\_linger\_time \leftarrow 2^{32} - 1$, $S_{best} \leftarrow \emptyset$;
if *Last used subflow* S_{last} *is available* **then**
  return S_{last} ;
end
foreach *subflow* $S_n \in S$ **do**
  $linger\_time \leftarrow \frac{wmem}{pace}$;
  if $linger\_time < best\_linger\_time$ **then**
    $S_{best} \leftarrow S_n$;
    $best\_linger\_time \leftarrow linger\_time$;
  end
end
return S_{best} ;

Figure 2. Default scheduler Algorithm.



(a) Subflow selection method



(b) Common Bottleneck

Figure 3. Default scheduler behavior and common bottleneck.

where Pace rate ($pace$) is the available transfer volume per second, and write memory data ($wmem$) is the value of the queued data volume in the subflow send buffer. Because $pace$ is dynamically adjusted with Round Trip Time (RTT) and congestion control, it can prevent head-of-line blocking while maintaining high throughput at all subflows. However, the default scheduler has one deficiency in full-mesh connection. If there are multiple subflows sending packets to the same destination and there is a common bottleneck link on the route, packets can cause buffering delays at the bottleneck link (Figure 3(b)). TCP congestion control will then back off from injecting new traffic, causing subflows to the same destination to interfere with each other, resulting in a long time before a new packet is injected on interfering flows. This problem is most likely to occur in TCP variants' congestion control where many packets are continuously sent, such as CUBIC.

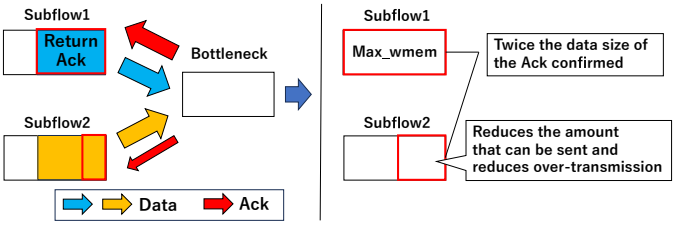


Figure 4. Buffer limitation method.

B. Bottleneck aware path scheduler

To solve the aforementioned problem, there is a need to limit the amount of packets injected on each subflow before too much queueing builds up at bottlenecks. A Bottleneck aware path scheduler is a scheduler with a limit on the amount of buffer used by each subflow, in addition to the short RTT mechanism of the default scheduler. The upper limit of the buffer used (max_wmem) is initially set to the amount of data that can be transmitted during 500 ms at the post-connection pace rate (ini_pace) (Eq:(2)). The reason for setting the initial maximum buffer amount to 500 ms of the pace rate is to ensure that all subflows do not slow down too much immediately after the start of the connection, as it is not known whether there are shared bottleneck links before transmission yet. Subsequently, if the smooth RTT ($sRTT$) is less than twice the minimum RTT (min_RTT), then twice the amount of data (snd_data) returned in ACK out of the data size transmitted between the previous and current subflow selection and the previous max_wmem are compared and the larger one is adopted (Eq:(3)). If $sRTT$ exceeds twice min_RTT and then returns to less than twice that value again, the smaller amount of data that can be transmitted per second with ini_pace or the previous max_wmem is adopted (Eq:(4)). By keeping max_wmem when $sRTT$ is more than twice min_RTT and updating it when $sRTT$ becomes less than twice again, the buffer size can be optimized while maintaining the throughput of each subflow.

- Initialization

$$max\_wmem = ini\_pace * 0.5\,[\text{byte}] \tag{2}$$

- If $sRTT < 2 * min\_RTT$

$$max\_wmem = max(max\_wmem, 2 * snd\_data)\,[\text{byte}] \tag{3}$$

- If $sRTT >= 2 * min\_RTT \rightarrow sRTT < 2 * min\_RTT$

$$max\_wmem = min(max\_wmem, ini\_pace * 1)\,[\text{byte}] \tag{4}$$

Iterating equations (3) and (4) during the transport session prevents excessive transmission, leading to a situation where too many packets accumulate in the subflow's buffer. This allows subflows with the same destination address to use paths that do not harm each other while satisfying the bandwidth of the bottleneck link (Figure 4).

V. CUBIC AND BBR TCP VARIANTS

The TCP protocol has evolved into different variants, implementing different congestion window adjustment schemes.

TCP protocol variants can be classified into delay and loss based congestion control schemes. Loss based TCP variants use packet loss as primary congestion indication signal, typically performing congestion window regulation as $cwnd_k = f(cwnd_{k-1})$, which is ACK reception paced. Most f functions follow an Additive Increase Multiplicative Decrease (AIMD) window adjustment scheme, with various increase and decrease parameters. In contrast, delay based TCP variants use queue delay information as the congestion indication signal, increasing/decreasing the window if the delay is small/large, respectively. Delay based congestion control does not suffer from packet loss undue window reduction due to random packet losses, as experienced in wireless links.

CUBIC TCP Congestion Avoidance: CUBIC TCP is a Loss-based TCP that has achieved widespread usage as the default TCP of the Linux operating system. During congestion avoidance, its congestion window is adjusted as follows (5):

$$\begin{aligned} AckRec: \quad cwnd_{k+1} &= C(t-K)^3 + Wmax \\ K &= (Wmax\frac{\beta}{C})^{1/3} \\ PktLoss: \quad cwnd_{k+1} &= \beta cwnd_k \\ Wmax &= cwnd_k \end{aligned} \tag{5}$$

where C is a scaling factor, Wmax is the cwnd value at time of packet loss detection, and t is the elapsed time since the last packet loss detection. The K parameter drives the CUBIC increase away from Wmax, whereas β tunes how quickly cwnd is reduced on packet loss. This adjustment strategy ensures that its $cwnd$ quickly recovers after a loss event.

BBR TCP Congestion Avoidance: BBR is a bandwidth delay product based TCP that has achieved widespread usage as one of available TCP variants in the Linux operating system. BBR uses measurements of a connection delivery rate and RTT to build a model that controls how fast data may be sent and the maximum amount of unacknowledged data in the pipe. Delivery rate is measured by keeping track of the number of acknowledged packets within a defined time frame. In addition, BBR uses a probing mechanism to determine the maximum delivery rate within multiple intervals. More specifically, BBR regulates the number of inflight packets to match the bandwidth delay product of the connection, or $BDP = BtlBw \times RTprop$, where $BtlBw$ is the bottleneck bandwidth of the connection, and $RTprop$ its propagation time, estimated as half of the connection RTT. These quantities are tracked during the lifetime of the connection, as per equations below (6):

$$\begin{aligned} rtt_t &= RTprop_t + \eta_t \\ \hat{RTprop} &= RTprop + min(\eta_t) \\ &= min(rtt_t) \forall t \in [T - W_R, T] \\ \hat{BtlBw} &= max(deliveryRate_t) \forall t \in [t - W_B, T] \end{aligned} \tag{6}$$

where η_t represents the noise of the queues along the path, W_R a running time window, of tens of seconds, and W_B a larger time window, of tens of RTTs. This adjustment strategy seeks to tune its $cwnd$ to a number of packets equivalent to the connection bandwidth delay product.

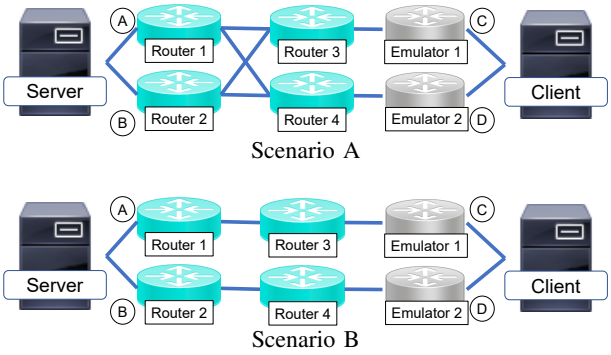


Figure 5. Experimental environment scenarios.

TABLE I. EXPERIMENTAL NETWORK SETTINGS

Element	Value
Video size	225 MBytes
Video rate	5.24 Mb/s
Playout time	6 mins
Video Codec	H264 MPEG-4 AVC
MPTCP variants	BBR, CUBIC
MPTCP schedulers	1. Default (Estimated shortest transmission time)
	2. Bottleneck aware

TABLE II. EXPERIMENTAL NETWORK SCENARIOS

Scenario	Emulator (BW, Packets Loss, Delay)
A: Fullmesh	A-1...BW: 3Mb/s, Loss: 0.1%, Delay: 60ms
	A-2...BW: 3Mb/s, Loss: 0.1%, Delay: 120ms
B: Parallel	BW: 3Mbps, Loss: 0.1%, Delay: 60ms

VI. Video Streaming Performance

Figure 5 describes the network testbed used for emulating network paths with wired links. An HTTP Nginx video server is connected to two L3 switches. In order to support multiple network scenarios, the L3 switches can be directly connected to another router, at which a client is connected. In this paper, the emulator boxes are used to vary each path RTT. We use two topology scenarios: a cross path scenario, where routers' cross-connections produce paths with common bottlenecks; and a parallel path scenario, where paths do not share common bottlenecks. These simple topologies and isolated traffic allow us to better understand the impact of differential delays on TCP variant's performance vis-a-vis path selection properties of path schedulers.

Application and network scenarios are described in Tables I and II, respectively. Video settings are typical of a video stream, with video playout rate of 5.24 Mb/s, and content size short enough to run multiple streaming trials within a short period of time. Four network scenarios are used (Figure 5). i) Two scenarios emulate fullmesh four path subflows, with short (60msec) and long (120msec) propagation delays; The four subflows share bottlenecks at routers close to the video server; ii) Two parallel path scenarios, for short and long propagation delays on two-path subflows, not sharing any bottlenecks. The fullmesh scenarios are used to expose the default scheduler's inadequacy as compared with our proposed bottleneck aware scheduler, whereas the parallel scenarios are meant to show "no harm" of a bottleneck aware scheduler when bottlenecks

are absent. Emulator boxes are tuned to generate multiple path network latency conditions. Path latency directly impacts the default path scheduler, as it gives preference to paths with shorter RTTs. Performance measures are:

- **Picture discards:** number of frames discarded by the video decoder.
- **Buffer underflow:** number of buffer underflow events at video client buffer.
- **Out-of-order packets:** Total number of out-of-order packets during each video streaming session.
- **subflow throughput:** TCP throughput of each subflow.

A. Fullmesh Scenarios

Scenarios A force the sharing of bottlenecks between the four paths available to MPTCP streaming. Each path has a maximum 3Mbit bandwidth, 0.1% packet loss rate, and 60ms or 120ms RTT delays.

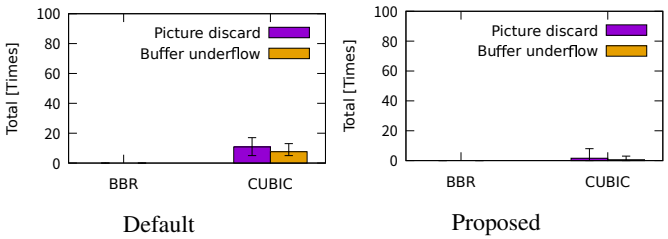
Figures 6 (a) and (b) show five average video streaming frame discard / buffer underflows, and the number of out-of-order packets, respectively for the short 60msec delay scenario. Notice the significant performance improvement of the bottleneck aware scheduler on frame discard and buffer underflow application level performance for the CUBIC TCP variant as compared to the MPTCP default scheduler. Interestingly enough, the average number of out-of-order packets does not seem to change significantly across schedulers. This seems to suggest that more out-of-order packets are concentrated on specific paths, reducing their impact on application level performance.

Figures 6 (c) and (d) show a single streaming trial of BBR and CUBIC, respectively, for the short 60msec delay scenario. The BBR TCP variant shows little throughput dynamic changes between schedulers. However, the CUBIC throughput seems to have become much more stable, showing an even throughput across all paths available. Throughput stability positively impacts video quality, as each path data reception and frame reassembly becomes more predictable.

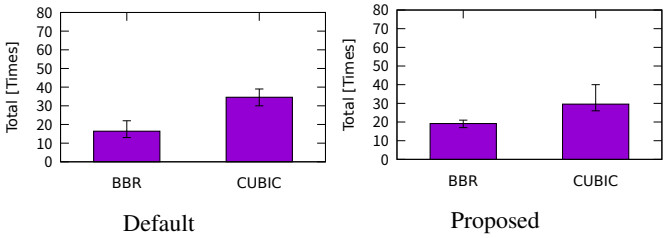
Figures 7 (a) and (b) show five average video streaming frame discard / buffer underflow, and the number of out-of-order packets, respectively, for the long 120msec delay scenario. Although frame discard and buffer underflow have become worse than the short delay scenario when the default scheduler is used, these application performance measures show the same qualitative benefits of the short delay scenario when using our proposed bottleneck aware scheduler. Figures 7 (c) and (d) show the same throughput stability benefits of our scheduler for CUBIC TCP variant.

B. No-cross-link Scenarios

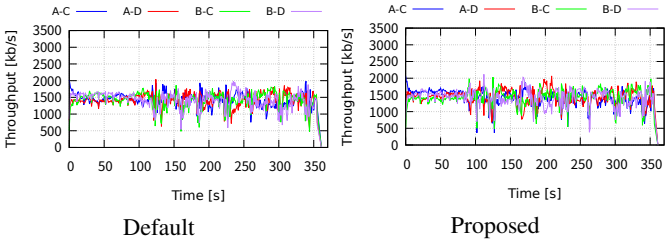
Figures 8 (a) and (b) show five average video streaming frame discard / buffer underflows, and the number of out-of-order packets, respectively, for the short 60msec delay scenario. Notice similar performance between the default and proposed the bottleneck aware schedulers, with the later still improving performance when CUBIC TCP variant is used.



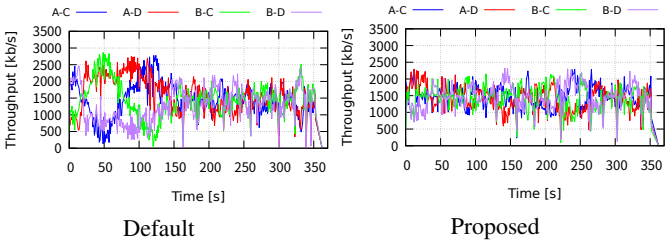
(a) Picture discard and Buffer underflow



(b) Out-of-order



(c) BBR throughput



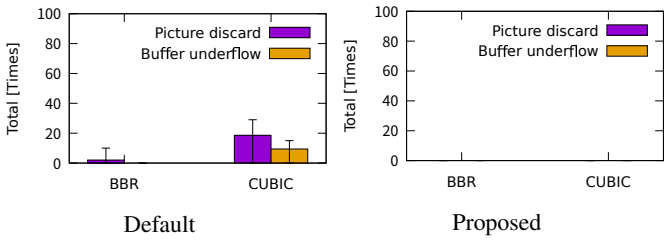
(d) CUBIC throughput

Figure 6. A-1 - Fullmesh Video Performance (delay 60ms.)

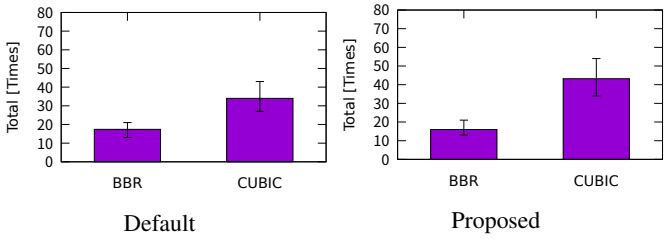
Figures 8 (c) and (d) show a single streaming trial of BBR and CUBIC, respectively, for the short 60msec delay scenario. Throughput dynamics are stable and similar for both schedulers. Hence, in the absence of shared bottlenecks, the proposed scheduler does not disturb throughput stability. Similar qualitative results are obtained for the long 120msec delay scenario, omitted for space's sake. These results verify that the proposed scheduler does not perform any worse than the default scheduler in the absence of shared path bottlenecks.

VII. Conclusion and Future Work

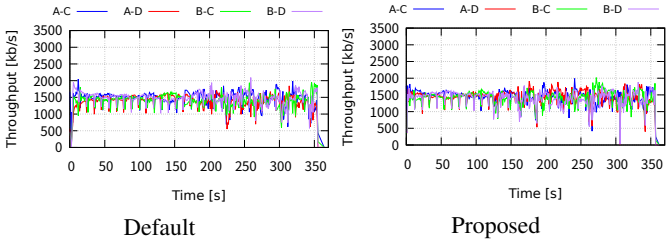
In this paper, we have studied multipath video streaming over fullmesh path scenarios. We have characterized the problem of path interference within a same video stream, demonstrating video application degradation when the default path scheduler is used. We have also proposed a bottleneck aware scheduler, where usage of paths containing a common bottleneck is controlled so as not to interfere negatively with each other. We have shown that the proposed scheduler not



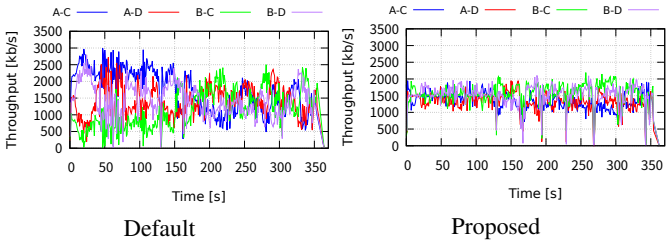
(a) Picture discard and Buffer underflow



(b) Out-of-order

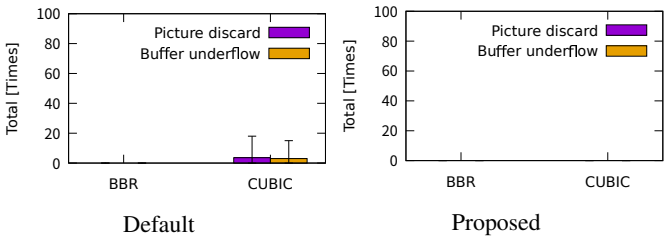


(c) BBR throughput

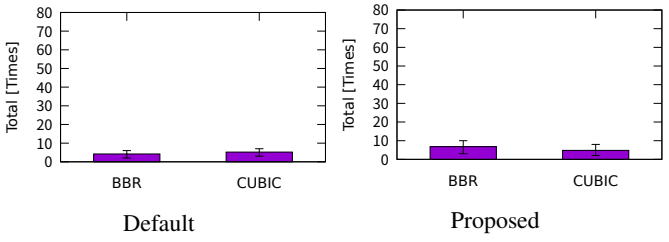


(d) CUBIC throughput

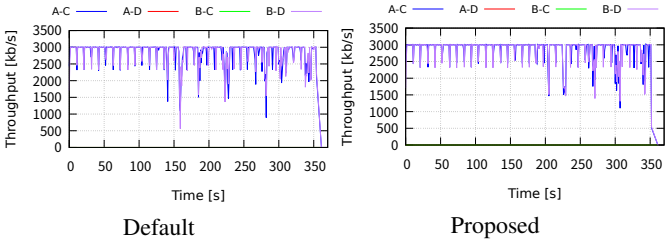
Figure 7. A-2 - Fullmesh Video Performance (delay 120ms).



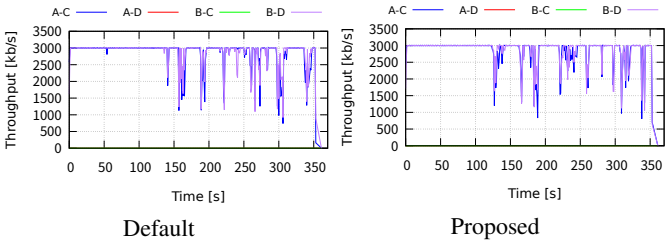
(a) Picture discard and Buffer underflow



(b) Out-of-order



(c) BBR throughput



(d) CUBIC throughput

Figure 8. B - Parallel Video Performance (delay 60ms).

only delivers better video application performance, but also helps ”stabilize” network throughput performance at each path. We have also shown that the proposed path scheduler does not cause performance degradation when paths do not share a bottleneck, when compared to the MPTCP default scheduler.

We are currently investigating the performance of our proposed scheduler on satellite access links.

ACKNOWLEDGMENTS

Work supported by JSPS KAKENHI Grant #24K03045.

REFERENCES

[1] I. Rhee, L. Xu, and S. Ha, “CUBIC for Fast Long-Distance Networks,” Internet Draft, draft-rhee-tcpm-ctcp-02, August 2008.

[2] N. Cardwell, Y. Cheng, I. Swett, and V. Jacobson, “BBR Congestion Control,” *IETF draft-cardwell-iccrg-bbr-congestion-control-01*, November 2021.

[3] M. R. Palash and K. Chen, “MPWiFi: Synergizing MPTCP Based Simultaneous Multipath Access and WiFi Network Performance,” IEEE Transactions on Mobile Computing, vol. 19, no. 1, pp. 142-158, Jan. 2020

[4] M. Amend, V. Rakocevic, and J. Habermann, “Cost optimized multipath scheduling in 5G for Video-on-Demand traffic,” In Proc. of IEEE Wireless Communications and Networking Conference - WCNC 21, pp. 1-6, March 2021.

[5] M. F. Imaduddin, A. G. Putrada, and S. A. Karimah, “Multipath TCP Scheduling Performance Analysis and Congestion Control on Video Streaming on the MPTCP Network,” In Proc. of Intern. Conference on Software Engineering & Computer Systems and 4th Intern. Conference on Computational Science and Information Management - ICSECS-ICOCSIM, pp. 562-567, August 2021.

[6] Y. Xing et al., “A Low-Latency MPTCP Scheduler for Live Video Streaming in Mobile Networks,” IEEE Transactions on Wireless Communications, vol. 20, no. 11, pp. 7230-7242, Nov. 2021.

[7] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, “Architectural Guidelines for Multipath TCP Development,” IETF RFC 6182, 2011.

[8] S. Nagayama, D. Cavendish, D. Nobayashi, and T. Ikenaga, “Path Switching Schedulers for MPTCP Streaming Video,” IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), August 2019, pp. 1-6.

[9] J. Zhao et al., “MPTCP+: Enhancing Adaptive HTTP Video Streaming over Multipath,” *Proceedings of IEEE/ACM International Symposium on Quality of Service (IWQoS)*, pp. 1-6, June 2020.

[10] B. Kimura et al., “Interpath Contention in MultiPath TCP Disjoint Paths,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1387-1400, August 2019.