Multipath TCP Packet Scheduling for Streaming Video

Ryota Matsufuji, Dirceu Cavendish, Kazumi Kumazoe, Daiki Nobayashi, Takeshi Ikenaga
Department of Computer Science and Electronics
Kyushu Institute of Technology
Fukuoka, Japan
e-mail: {q349428r@mail, cavendish@ndrc, kuma@ndrc, nova@ecs, ike@ecs}.kyutech.ac.jp

***Abstract*—Video streaming has become the major source of Internet traffic nowadays. Considering that content delivery network providers have adopted Video over Hypertext Transfer Protocol/Transmission Control Protocol (HTTP/TCP) as the preferred protocol stack for video streaming, understanding TCP performance in transporting video streams has become paramount. Recently, multipath transport protocols have allowed video streaming over multiple paths to become a reality. In this paper, we propose packet scheduling disciplines for injecting video stream packets into multiple paths at the video server. We study video streaming performance when subjected to these schedulers in conjunction with current TCP variants. We utilize network performance measures, as well as video quality metrics, to characterize the performance and interaction between network and application layers of video streams for various network scenarios.**

***Keywords*—Video streaming; high speed networks; TCP congestion control; Multipath TCP; Packet retransmissions; Packet loss.**

I. INTRODUCTION

Transmission control protocol (TCP) is the dominant transport protocol of the Internet, providing reliable data transmission for the large majority of applications. For data applications, the perceived quality of service is the total transport time of a given file. For real time (streaming) applications, the perceived quality of experience involves not only the total transport time, but also the amount of data discarded at the client due to excessive transport delays, as well as rendering stalls due to the lack of timely data. Transport delays and data starvation depend on how TCP handles flow control and packet retransmissions. Therefore, video streaming user experience depends heavily on TCP performance.

TCP protocol interacts with video application in non trivial ways. Widely used video codecs, such as H-264, use compression algorithms that result in variable bit rates along the playout time. In addition, TCP has to cope with variable network bandwidth along the transmission path. Network bandwidth variability is particularly wide over paths with wireless access links of today, where multiple transmission modes are used to maintain steady packet error rate under varying interference conditions. As the video playout rate and network bandwidth are independent, it is the task of the transport protocol to provide a timely delivery of video data so as to support a smooth playout experience.

Recently, multipath transport has allowed video streamed over multiple IP interfaces and network paths. Multipath streaming not only augments aggregated bandwidth, but also increases reliability at the transport level session even when a specific radio link coverage gets compromised. An important issue in multipath transport is the path (sub-flow) selection; a packet scheduler is needed to split traffic to be injected on a packet by packet basis. For video streaming applications, head of line blocking may cause incomplete or late frames to be discarded at the receiver, as well as stream stalling. In this work, we propose a couple of path schedulers and evaluate video streaming performance under these schedulers. To the best of our knowledge, there has not been a study of path selection mechanisms' performance of multipath video streaming in the literature.

The material is organized as follows. Related work discussion is provided on Section II. Section III describes video streaming over TCP system. Section IV introduces the TCP variants addressed in this paper, as well as Multipath TCP and path schedulers used to support multipath transport. Section V addresses multiple path video delivery performance evaluation for each TCP variant. Section VI addresses directions we are pursuing as follow up to this work.

II. RELATED WORK

Although multipath transport studies are plenty in the literature, there has been few prior work on video performance over multiple paths [5] [11] [15]. In our previous work [10], Matsufuji et al. have evaluated multipath video streaming performance when widely deployed TCP variants are used in each path. Regarding multipath schedulers, there has been even less research activity. Yan et al. [16] propose a path selection mechanism based on estimated sub-flow capacity. Their evaluation is centered on throughput performance, as well as reducing packet retransmissions. Yan et al. [2] present a modelling of multipath transport in which they explain empirical evaluations of the impact of selecting a first subflow in throughput performance. Hwang et al. [8] propose a blocking scheme of a slow path when delay difference between paths is large, in order to improve data transport completion time on short lived flows. Finally, Ferlin et al. [6] introduces a path selection scheme based on a predictor of the head-of-line blocking of a given path. They carry out emulation experiments with their scheduler against the minimum rtt default scheduler, in transporting bulk data, Web transactions, and Constant Bit Rate (CBR) traffic, with figure of merits of goodput, completion time, and packet delays, respectively.

In contrast, our work seeks to propose and evaluate multipath path scheduling mechanisms and their impact on the quality of video streams. Previously [10], we have evaluated multipath video streaming using standard MPTCP path selection scheduler. In this work, we evaluate two new path sched-

 ISBN: 978-1-61208-574-6

uler proposals. For performance evaluation, we use widely deployed TCP variants on open source network experiments over WiFi access links. The use of widely deployed TCP variants is motivated by the fact that path selection is constrained by the availability and size of congestion window controlled by TCP variants on each path.

III. Video Streaming over TCP

Video streaming over HTTP/TCP involves an HTTP server, where video files are made available for streaming upon HTTP requests, and a video client, which places HTTP requests to the server over the Internet, for video streaming. Figure 1 illustrates video streaming components.

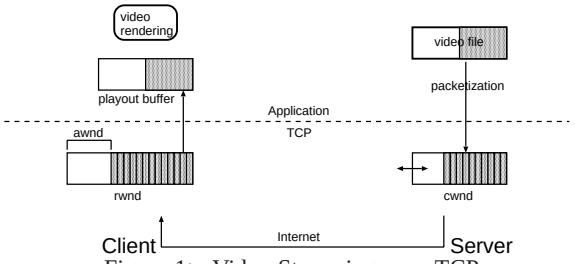


Figure 1: Video Streaming over TCP

An HTTP server stores encoded video files, available upon HTTP requests. Once a request is placed, a TCP sender is instantiated to transmit packetized data to the client machine. At TCP transport layer, a congestion window is used for flow controlling the amount of data injected into the network. The size of the congestion window, $cwnd$, is adjusted dynamically, according to the level of congestion in the network, as well as the space available for data storage, $awnd$, at the TCP client receiver buffer. Congestion window space is freed only when data packets are acknowledged by the receiver, so that lost packets are retransmitted by the TCP layer. At the client side, in addition to acknowledging arriving packets, TCP receiver sends back its current available space $awnd$, so that at the sender side, $cwnd \leq awnd$ at all times. At the client application layer, a video player extracts data from a playout buffer, filled with packets delivered by TCP receiver from its buffer. The playout buffer is used to smooth out variable data arrival rate.

A. Interaction between Video streaming and TCP

At the server side, HTTP server retrieves data into the TCP sender buffer according with $cwnd$ size. Hence, the injection rate of video data into the TCP buffer is different than the video variable encoding rate. In addition, TCP throughput performance is affected by the round trip time of the TCP session. This is a direct consequence of the congestion window mechanism of TCP, where only up to a $cwnd$ worth of bytes can be delivered without acknowledgements. Hence, for a fixed $cwnd$ size, from the sending of the first packet until the first acknowledgement arrives, a TCP session throughput is capped at $cwnd/rtt$. For each TCP congestion avoidance scheme, the size of the congestion window is computed by a specific algorithm at time of packet acknowledgement reception by the TCP source. However, for all schemes, the size of the congestion window is capped by the available TCP receiver space $awnd$ sent back from the TCP client.

At the client side, the video data is retrieved by the video player into a playout buffer, and delivered to the video renderer. Playout buffer may underflow, if TCP receiver window empties out. On the other hand, playout buffer overflow does not occur, since the player will not pull more data into the playout buffer than it can handle.

In summary, video data packets are injected into the network only if space is available at the TCP congestion window. Arriving packets at the client are stored at the TCP receiver buffer, and extracted by the video playout client at the video nominal playout rate.

IV. Anatomy of transmission control protocol

TCP protocols fall into two categories, delay and loss based. Advanced loss based TCP protocols use packet loss as primary congestion indication signal, performing window regulation as $cwnd_k = f(cwnd_{k-1})$, being ack reception paced. Most f functions follow an Additive Increase Multiplicative Decrease strategy, with various increase and decrease parameters. TCP NewReno [1] and Cubic [13] are examples of additive increase multiplicative decrease (AIMD) strategies. Delay based TCP protocols, on the other hand, use queue delay information as the congestion indication signal, increasing/decreasing the window if the delay is small/large, respectively. Compound [14], Capacity and Congestion Probing (CCP) [3] and Capacity Congestion Plus Derivative (CCPD) [4] are examples of delay based protocols.

Most TCP variants follow TCP Reno phase framework: slow start, congestion avoidance, fast retransmit, and fast recovery.

- **Slow Start(SS):** This is the initial phase of a TCP session. In this phase, for each acknowledgement received, two more packets are allowed into the network. Hence, congestion window $cwnd$ is roughly doubled at each round trip time. Notice that $cwnd$ size can only increase in this phase. So, there is no flow control of the traffic into the network. This phase ends when $cwnd$ size reaches a large value, dictated by $ssthresh$ parameter, or when the first packet loss is detected, whichever comes first. All widely used TCP variants use slow start except Cubic [13].
- **Congestion Avoidance(CA):** This phase is entered when the TCP sender detects a packet loss, or the $cwnd$ size reaches the target upper size $ssthresh$ (slow start threshold). The sender controls the $cwnd$ size to avoid path congestion. Each TCP variant has a different method of $cwnd$ size adjustment.
- **Fast Retransmit and fast recovery(FR):** The purpose of this phase is to freeze all $cwnd$ size adjustments in order to take care of retransmissions of lost packets.

For TCP variants widely used today, congestion avoidance phase is sharply different. We will be introducing specific TCP variants' congestion avoidance phase shortly.

A. Multipath TCP

Multipath TCP (MPTCP) is a transport layer protocol, currently being evaluated by IETF, which makes possible data transport over multiple TCP sessions [7]. The key idea is to

make multipath transport transparent to upper layers, hence presenting a single TCP socket to applications. Under the hood, MPTCP works with TCP variants, which are unaware of the multipath nature of the overall transport session. To accomplish that, MPTCP supports a packet scheduler that extracts packets from the MPTCP socket exposed to applications, and injects them into TCP sockets belonging to a "sub-flow" defined by a single path TCP session. MPTCP transport architecture is represented in Figure 2.

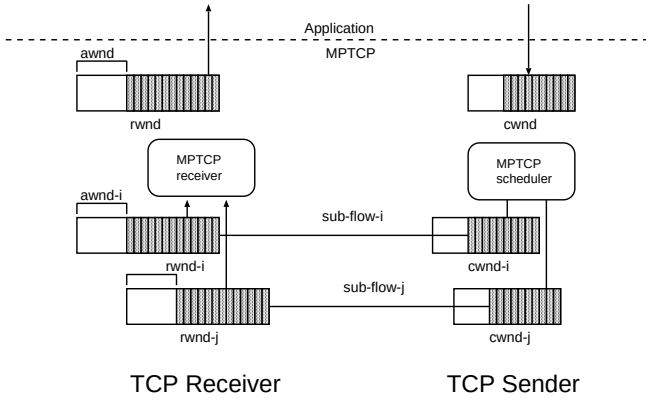


Figure 2: MPTCP Architecture

MPTCP packet scheduler works in two different configuration modes: uncoupled, and coupled. In uncoupled mode, each sub-flow congestion window $cwnd$ is adjusted independently. In coupled mode, MPTCP couples the congestion control of the sub-flows, by adjusting the congestion window $cwnd_k$ of a sub-flow k according with parameters of all sub-flows. Although there are several coupled mechanisms, we focus on Linked Increase Algorithm (LIA) [12] and Opportunistic Linked Increase Algorithm (OLIA) [9]. In both cases, a MPTCP scheduler selects a sub-flow for packet injection according to some criteria among all sub-flows with large enough $cwnd$ to allow packet injection.

Multipath Scheduling: MPTCP scheduler has the role of selecting which sub-flow to inject packets into the network. The default strategy is to select a path with shortest current packet delay. Here, we introduce two other path selection and packet injection mechanisms.

- **Shortest Packet Delay(SPD):** In shortest packet delay, the scheduler first rules out any path for which there is no space in its sub-flow congestion window (cwnd). Among the surviving paths, the scheduler then selects the path with small smooth round trip time (rtt). Smooth rtt is computed as an average rtt of recent packets transmitted at that sub-flow. Since each sub-flow already keeps track of its smooth rtt, this quantity is readily available at every sub-flow.
- **Largest packet credits(LPC):** Among the sub-flows with space in their cwnd, this scheduler selects the one with largest available space. Available space is the number of packets allowed by cwnd size minus the packets that have not been acknowledged yet.
- **Largest Estimated Throughput(LET):** In this case, among the sub-flows with large enough cwnd to accommodate new packets, the scheduler estimates the throughput of each sub-flow and selects the one with largest throughput.

The rationale for the proposed schedulers is as follows. LPC addresses the path scenario in which a large rtt path has plenty of bandwidth. In default scheduler, this path may be less preferred due to its large rtt, regardless of having plenty of bandwidth for the video stream. LET addresses the scenario of a short path with plenty of bandwidth. The default scheduler may select this path due to its short rtt. However, if the short rtt has a smaller cwnd, LET will divert traffic away from this path, whereas default scheduler will continue to inject traffic through it. In summary, a significant difference between these two proposed schedulers and the default scheduler is that path selection relies on path characteristics that are more dynamic (cwnd, in flight packet count) than packet delay (rtt).

B. Linked Increase Congestion Control

Link Increase Algorithm [12] couples the congestion control algorithms of different sub-flows by linking their congestion window increasing functions, while adopting the standard halving of $cwnd$ window when a packet loss is detected. More specifically, LIA cwnd adjustment scheme is as per (1):

$$AckRec: cwnd^i_{k+1} = cwnd^i_k + min(\frac{\alpha B_{ack} Mss^i}{\sum_0^n cwnd^p}, \frac{B_{ack} Mss^i}{cwnd^i})$$
$$PktLoss: cwnd^i_{k+1} = \frac{cwnd^i_k}{2} \quad (1)$$

where α is a parameter regulating the aggressiveness of the protocol, B_{ack} is the number of acknowledged bytes, Mss^i is the maximum segment size of sub-flow i, and n is the number of sub-flows. Equation (1) adopts $cwnd$ in bytes, rather than in packets (MSS), in contrast with TCP variants equations to be described shortly, because here we have the possibility of diverse MSSs on different sub-flows. However, the general idea is to increase $cwnd$ in increments that depend on $cwnd$ size of all sub-flows, for fairness, but no more than a single TCP Reno flow. The min operator in the increase adjustment guarantees that the increase is at most the same as if MPTCP was running on a single TCP Reno sub-flow. Therefore, in practical terms, each LIA sub-flow increases $cwnd$ at a slower pace than TCP Reno, still cutting $cwnd$ in half at each packet loss.

C. Opportunistic Linked Increase Congestion Control

Opportunistic Link Increase Algorithm [9] also couples the congestion control algorithms of different sub-flows, but with the increase based on the quality of paths. More specifically, OLIA cwnd adjustment scheme is as per (2):

$$AckRec: cwnd^i_{k+1} = cwnd^i_k + \frac{\frac{cwnd^i}{(rtt^i)^2}}{(\sum_0^n \frac{cwnd^p}{rtt^p})^2} + \frac{\alpha^i}{cwnd^i},$$
$$PktLoss: cwnd^i_{k+1} = \frac{cwnd^i_k}{2} \quad (2)$$

where α is a positive parameter for all paths. The general idea is to tune $cwnd$ to an optimal congestion balancing point (in the Pareto optimal sense). In practical terms, at each OLIA sub-flow increases $cwnd$ at a pace related to the ratio of its rtt and rtt of other subflows, still cutting $cwnd$ in half at each packet loss.

 ISBN: 978-1-61208-574-6

D. Cubic TCP Congestion Avoidance

TCP Cubic 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 adjustment scheme is:

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

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 (cwnd reduction). The rationale for these equations is simple. Cubic remembers the cwnd value at time of packet loss detection - Wmax, when a sharp cwnd reduction is enacted, tuned by parameter β. After that, cwnd is increased according to a cubic function, whose speed of increase is dictated by two factors: i) how long it has been since the previous packet loss detection, the longer the faster ramp up; ii) how large the cwnd size was at time of packet loss detection, the smaller the faster ramp up. The shape of Cubic cwnd dynamics is distinctive, clearly showing its cubic nature. Notice that upon random loss, Cubic strives to return cwnd to the value prior to loss detection quickly, for small cwnd sizes.

E. Compound TCP Congestion Avoidance

Compound TCP is the TCP of choice for most deployed Wintel machines. It implements a hybrid loss/delay based congestion avoidance scheme, by adding a delay congestion window dwnd to the congestion window of NewReno [14]. Compound TCP cwnd adjustment is as per (4):

$$\begin{aligned} AckRec: \quad cwnd_{k+1} &= cwnd_k + \frac{1}{cwnd_k + dwnd_k} \\ PktLoss: \quad cwnd_{k+1} &= cwnd_k + \frac{1}{cwnd_k} \end{aligned} \tag{4}$$

where the delay component is computed as:

$$\begin{aligned} AckRec: dwnd_{k+1} = dwnd_k + \alpha dwnd_k^K - 1, & \text{ if } diff < \gamma \\ dwnd_k - \eta diff, & \text{ if } diff \geq \gamma \\ PktLoss: dwnd_{k+1} = dwnd_k(1-\beta) - \frac{cwnd_k}{2} & \end{aligned} \tag{5}$$

where $diff$ is an estimated number of backlogged packets, γ is a threshold parameter which drives congestion detection sensitivity, and α, β, η and K are parameters chosen as a tradeoff between responsiveness, smoothness, and scalability.

Compound TCP dynamics is dominated by its loss based component, presenting a slow responsiveness to network available bandwidth variations, which may cause playout buffer underflows.

F. Capacity and Congestion Probing TCP

TCP CCP was our first proposal of a delay based congestion avoidance scheme based on solid control theoretical approach. The cwnd size is adjusted according to a proportional controller control law. The cwnd adjustment scheme is called at every acknowledgement reception, and may result in either window increase or decrease. In addition, packet loss does not trigger any special cwnd adjustment. CCP cwnd adjustment scheme is as per (6):

$$cwnd_k = \frac{[Kp(B - x_k) - in\_flight\_segs_k]}{2} \quad 0 \leq Kp \tag{6}$$

where Kp is a proportional gain, B is an estimated storage capacity of the TCP session path, or virtual buffer size, x_k is the level of occupancy of the virtual buffer, or estimated packet backlog, and in_flight_segs is the number of segments in flight (unacknowledged). Typically, CCP cwnd dynamics exhibit a dampened oscillation towards a given cwnd size, upon cross traffic activity. Notice that $cwnd_k$ does not depend on previous cwnd sizes, as with the other TCP variants. This fact guarantees a fast responsiveness to network bandwidth variations.

V. Video Streaming Performance of Multipath Schedulers

Figure 3 describes the network testbed used for emulating a network path with wireless access link. An HTTP video server is connected to two access switches, which are connected to a link emulator, used to adjust path delay and inject controlled random packet loss. A VLC client machine is connected to two Access Points, a 802.11a and 802.11g, on different bands (5GHz and 2.4GHz, respectively). All wired links are 1Gbps. No cross traffic is considered, as this would make it difficult to isolate the impact of TCP congestion avoidance schemes on video streaming performance. The simple topology and isolated traffic allows us to better understand the impact of differential delays on streaming performance.

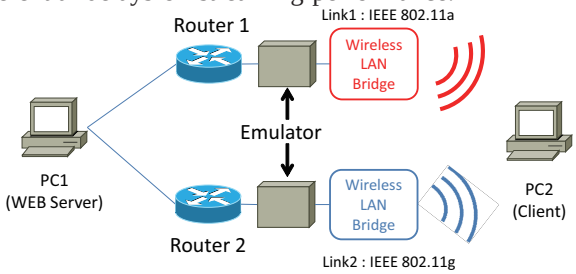


Figure 3: Video Streaming Emulation Network

TCP variants used are: Cubic, Compound, CCP, LIA and OLIA. Performance is evaluated for various round trip time path scenarios, as per Table I.

Table I: EXPERIMENTAL NETWORK SETTINGS

Element	Value
Video size	409Mbytes
Video rate	5.24Mbps
Playout time	10mins 24 secs
Encoding	MPEG-4
Video Codec	H.264 MPEG-4 AVC
Network Delay (RTT)	100, 200 msecs
TCP variants	Cubic, Compound, CCP, LIA, OLIA

The VLC client is attached to the network via a WiFi link. Iperf is used to measure the available wireless link bandwidth. UDP traffic injection experiments show that each wireless interface is limited to 5Mbps download speeds, which is lower than the video nominal playout rate of 5.24Mbps. Packet loss is hence induced only by the wireless link, and is reflected in the number of TCP packet retransmissions.

 ISBN: 978-1-61208-574-6

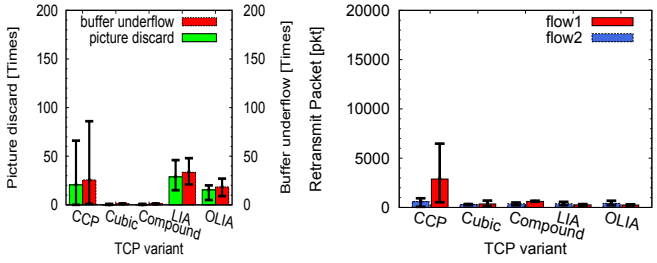
Performance measures adopted, in order of priority, are:

- **Picture discards:** number of frames discarded by the video decoder. This measure defines the number of frames skipped by the video rendered at the client side.
- **Buffer underflow:** number of buffer underflow events at video client buffer. This measure defines the number of "catch up" events, where the video freezes and then resumes at a faster rate until all late frames have been played out.
- **Packet retransmissions:** number of packets retransmitted by TCP. This is a measure of how efficient the TCP variant is in transporting the video stream data. It is likely to impact video quality in large round trip time path conditions, where a single retransmission doubles network latency of packet data from an application perspective.

We organize our video streaming experimental results into the following sub-sessions: i) Equal path delay; ii) Differential path delay. Each data point in charts represents five trials. Results are reported as average and min/max deviation bars.

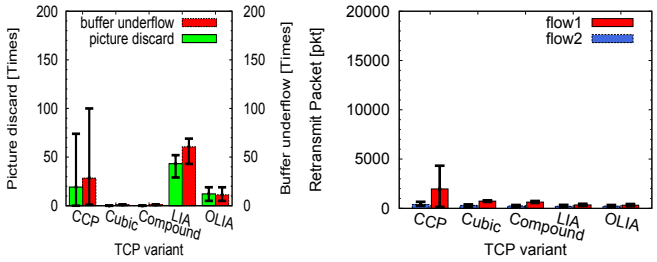
A. Equal Path Video Streaming Performance Evaluation

Figures 4, 5, and 6 report on video streaming throughput performance over equal paths delay of 100 msecs, for SPD, LPC, and LET schedulers, respectively. Notice first that under SPD, Cubic and Compound TCP variants deliver best video performance. In contrast, OLIA, CCP, and LIA deliver worse performance. When comparing SPD, LPC, and LET schedulers, video streaming performance under OLIA variants improves using LPC or LET schedulers, while it remains the same for Cubic and Compound TCP variants.



a) VLC performance b) TCP packets retransmitted

Figure 4: SPD Scheduler Streaming Perf.; rtts=100-100msecs

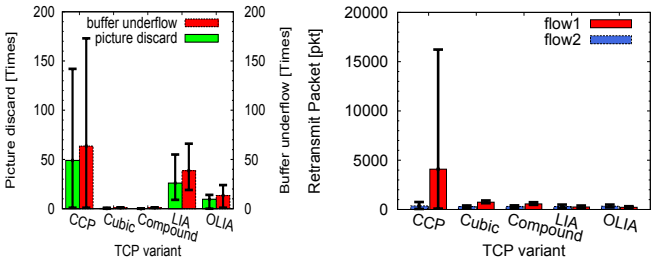


a) VLC performance b) TCP packets retransmitted

Figure 5: LPC Scheduler Streaming Perf.; rtts=100-100msecs

B. Differential Path Video Streaming Performance Evaluation

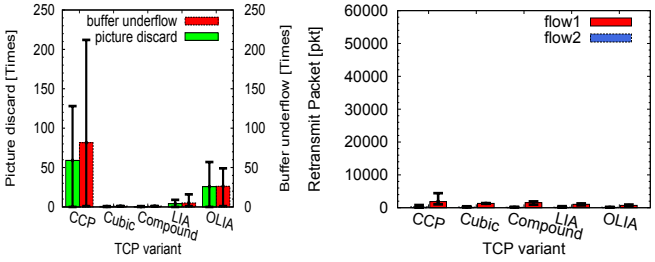
In these scenarios, default MPTCP scheduler tends to select the path with shorter delay, if $cwnd$ permits it. Only when TCP sender of the path with shorter delay happens to set its $cwnd$ to a very low value as compared with the longer path does MPTCP scheduler inject packets into the longer path.



a) VLC performance b) TCP packets retransmitted

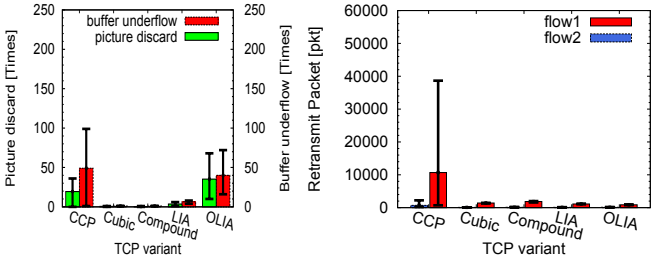
Figure 6: LET Scheduler Streaming Perf.; rtts=100-100msecs

Figures 7, 8, and 9 report on video streaming and TCP performance under two paths, the first path (802.11a) with a shorter 100msec delay, and the other (802.11g) with a longer 200msec delay, for SPD, LPC, and LET schedulers, respectively. Under default (SPD) scheduler, we continue to see better video performance under Cubic and Compound than CCP or coupled LIA and OLIA variants. When comparing schedulers, video streaming performance under CCP variant improves using LPC scheduler, it worsens for OLIA variant, and remains the same for Cubic and Compound TCP variants. CCP variant improvement comes at a cost of larger packet retransmissions, however.



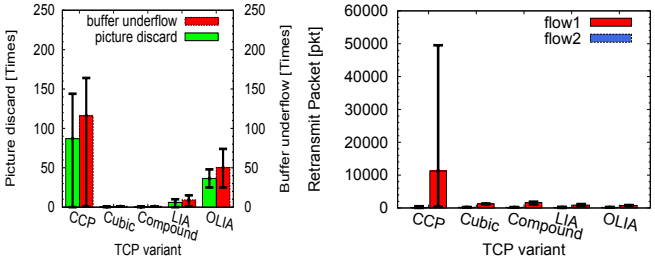
a) VLC performance b) TCP packets retransmitted

Figure 7: SPD Scheduler Streaming Perf.; rtts=100-200msecs



a) VLC performance b) TCP packets retransmitted

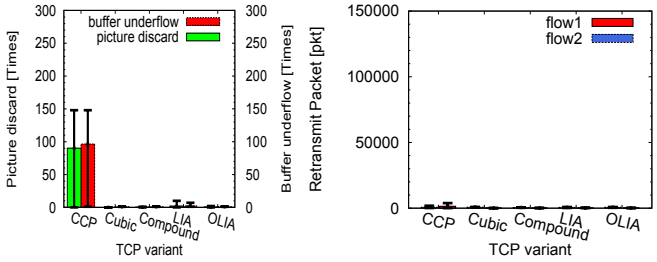
Figure 8: LPC Scheduler Streaming Perf.; rtts=100-200msecs



a) VLC performance b) TCP packets retransmitted

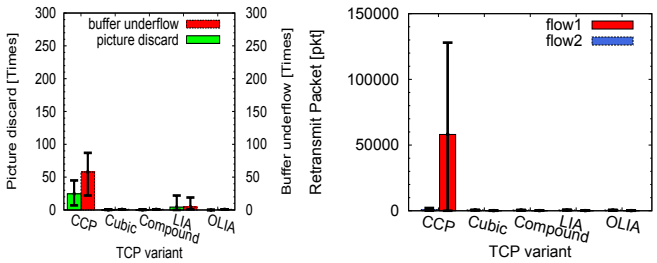
Figure 9: LET Scheduler Streaming Perf.; rtts=100-200msecs

Figures 10, 11, and 12 report on video streaming and TCP performance under two paths, the first path (802.11a) with a longer 200msec delay, and the other (802.11g) with a shorter 100msec delay, for SPD, LPC, and LET schedulers, respectively. Under default (SPD) scheduler, all TCP variants deliver similar video performance except CCP. When comparing path schedulers, video streaming performance under CCP variant improves using LPC scheduler, while it remains the same for the other TCP variants. Again, CCP video improvement comes at a cost of higher packet retransmissions.



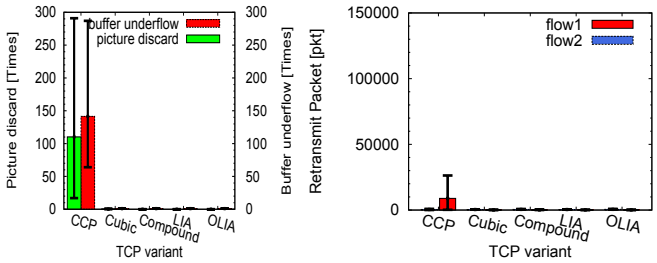
a) VLC performance b) TCP packets retransmitted

Figure 10: SPD Scheduler Streaming Perf.; rtts=200-100msecs



a) VLC performance b) TCP packets retransmitted

Figure 11: LPC Scheduler Streaming Perf.; rtts=200-100msecs



a) VLC performance b) TCP packets retransmitted

Figure 12: LET Scheduler Streaming Perf.; rtts=200-100msecs

In conclusion, Largest Packet Credit MPTCP scheduler improves video streaming performance of CCP variant overall, while having positive or no enhancement on the other TCP variants. We also detected that OLIA delivers better video experience than LIA coupled TCP variant across all path scenarios. We notice, however, that a two path transport scenario is constraining, in the sense that if one of the paths is blocked for transmission, for instance, due to some packet loss and small $cwnd$, all schedulers will select the other path, and hence will likely deliver the same performance. In our scenario, we traced a larger packet loss behavior of flow 2, which leads to different utilization of paths and performance if comparing 100-200msec delay scenario with 200-100msec delay scenario. Increasing the number of paths is a possibility, albeit such scenario may not be realistic in the near future.

VI. Conclusion and Future work

In this paper, we have proposed two new path schedulers and evaluated them on Multipath TCP transport of video streaming, using widely deployed TCP variants, as well as LIA and OLIA coupled TCP variants under consideration by IETF. We have characterized MPTCP performance with default and proposed path schedulers when transporting video streaming over two wireless network paths via open source experiments. Our experimental results show that injecting packets at the path with largest packet credits (cwnd - in flight packets) yields better video performance for OLIA coupled TCP variant and CCP. Cubic and Compound TCP variants deliver the same performance under all path schedulers studied. Hence, from a video performance viewpoint, either MPTCP in uncoupled mode or coupled with largest packet credit scheduler should be used. We are currently analyzing path schedulers' performance on more diverse multipath network scenarios.

Acknowledgments

This work is supported by JSPS KAKENHI Grant Number 16K00131.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," IETF RFC 2581, April 1999.

[2] B. Arzani et al., "Deconstructing MPTCP Performance," In Proceedings of IEEE 22nd ICNP, pp. 269-274, 2014.

[3] D. Cavendish, K. Kumazoe, M. Tsuru, Y. Oie, and M. Gerla, "Capacity and Congestion Probing: TCP Congestion Avoidance via Path Capacity and Storage Estimation," IEEE Second International Conference on Evolving Internet, best paper award, pp. 42-48, September 2010.

[4] D. Cavendish, H. Kuwahara, K. Kumazoe, M. Tsuru, and Y. Oie, "TCP Congestion Avoidance using Proportional plus Derivative Control," IARIA Third International Conference on Evolving Internet, best paper award, pp. 20-25, June 2011.

[5] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-Layer Scheduler for Video Streaming over MPTCP," ACM 7th International Conference on Multimedia Systems, May 10-13,2016, Article 7.

[6] S. Ferlin, et. al., "BLEST: Blocking Estimation-based MPTCP Scheduler for Heterogeneous Networks," In Proceedings of IFIP Networking Conference, pp. 431-439, 2016.

[7] A. Ford, et al., "Architectural Guidelines for Multipath TCP Development," IETF RFC 6182, 2011.

[8] J. Hwang and J. Yoo, "Packet Scheduling for Multipath TCP," IEEE 7th Int. Conference on Ubiquitous and Future Networks, pp.177-179, July 2015.

[9] R. Khalili, N. Gast, and J-Y Le Boudec, "MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution," IEEE/ACM Trans. on Networing, Vol. 21, No. 5, pp. 1651-1665, Aug. 2013.

[10] R. Matsufuji et al., "Performance Characterization of Streaming Video over Multipath TCP," IARIA 8th International Conference on Evolving Internet, pp. 42-47, November 2016.

[11] J-W. Park, R. P. Karrer, and J. Kim,, "TCP-Rome: A Transport-Layer Parallel Streaming Protocol for Real-Time Online Multimedia Environments," In Journal of Communications and Networks, Vol.13, No. 3, pp. 277-285, June 2011.

[12] C. Raiciu, M. Handly, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," IETF RFC 6356, 2011.

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

[14] M. Sridharan, K. Tan, D. Bansal, and D. Thaler, "Compound TCP: A New Congestion Control for High-Speed and Long Distance Networks," Internet Draft, draft-sridharan-tcpm-ctcp-02, November 2008.

[15] J. Wu, C. Yuen, B. Cheng, M. Wang, and J. Chen, "Streaming High-Quality Mobile Video with Multipath TCP in Heterogeneous Wireless Networks," IEEE Transactions on Mobile Computing, Vol.15, Issue 9, pp. 2345-2361, 2016.

[16] F. Yan, P. Amer, and N. Ekiz, "A Scheduler for Multipath TCP," In Proceedings of IEEE 22nd ICCCN, pp. 1-7, 2013.

 ISBN: 978-1-61208-574-6