Managing Path Switching in Multipath Video Streaming

Shinichi Nagayama, Dirceu Cavendish, Daiki Nobayashi, Takeshi Ikenaga
Department of Computer Science and Electronics
Kyushu Institute of Technology
Fukuoka, Japan
e-mail: {o108076s@mail}.kyutech.jp {cavendish@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 utilize 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 analyze the impact of path switching on multipath video streaming and network performance, and propose new schedulers which minimize the number of path switching. 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; TCP socket state; Multipath TCP; Packet retransmissions; Packet loss.

I. INTRODUCTION

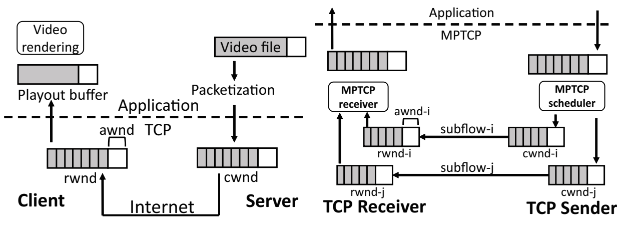
Transmission Control Protocol (TCP) is the dominant transport protocol of the Internet, providing reliable data transmission for the large majority of applications. For data applications, the perceived quality of service is the total transport time of a given file. For real time (streaming) applications, 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.

Recently, multipath transport has allowed video streams 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 path scheduler is needed to split traffic to be injected on a packet by packet basis onto available paths. 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 analyze the effect of path switching on the quality of video stream delivery. In addition, we propose path switch aware schedulers, which strive to minimize the number of path switches during a video stream delivery session while improving video performance. We show that, by selectively controlling path switching, video streaming performance improvements can be obtained for widely deployed TCP variants and network scenarios.

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. Section V analyzes path switching effects on video performance, and introduces our new path scheduling proposals, generically called sticky schedulers. Section VI addresses multiple path video delivery performance evaluation using default path scheduler vis a vis several sticky schedulers, for each TCP variant and multiple packet schedulers. Our empirical results show that most TCP variants deliver better video performance when sticky scheduling is utilized. Section VII 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 limited prior work on video performance over multiple paths [4] [14] [19]. Regarding multipath schedulers, there has been recent research activity, propelled by the availability of Multipath Transmission Control Protocol (MPTCP) transport stack. Most of them focus on specific sub-flow characterization to support smart path selection. For instance, Yan et al. [21] propose a path selection mechanism based on estimated sub-flow capacity. Their evaluation is centered on throughput performance, as well as reducing packet retransmissions. Hwang et al. [9] 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. Ferlin et al. [6] introduce 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 Round Trip Time (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. More recently, Kimura et al. [11] have shown throughput performance improvements on schedulers driven by path sending rate and window space, focusing on bulk data transfer applications. Also, Dong et al. [5] have proposed a path loss estimation approach to select paths subject to high and bulk loss rates. Although they have presented some video streaming experiments, they do not measure streaming performance from an application perspective. Xue et al. [20] has proposed a path scheduler based on prediction of the amount of data a path is able to transmit and evaluated it on simulated



(a) TCP (b) MPTCP

Figure 1: Video Streaming over TCP/MPTCP

network scenarios with respect to throughput performance. A different approach, at which different sub-flows are used for segregating prioritized packets of Augmented Reality/Virtual Reality streams has been proposed by Silva et al. [18]. Finally, Frommgen et al. [8] have shown that stale round trip time (rtt) information interferes with path selection of small streams such as HTTP traffic. The authors then propose an rtt probing and one way delay based path selection to improve latency and throughput performance of thin streams.

In contrast, our current work seeks multipath path scheduling principles that can be applied to different path schedulers to specifically improve the quality of video streams. Previously [12], we have proposed new Multipath TCP path schedulers based on dynamic path characteristics, such as congestion window space and estimated path throughput, and evaluated multipath video streaming using these proposed schedulers. Recently [13], we have also proposed to enhance path schedulers with TCP state information, such as whether a path is in fast retransmit and fast recovery state, to improve video quality in lossy network scenarios. In this work, we propose one more principle to path selection, the minimization of path switching. We evaluate new path schedulers, called sticky schedulers, on video stream applications using widely deployed TCP variants on open source network experiments over WiFi an wired access links.

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 (a) illustrates video streaming components.

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, the HTTP server retrieves data into the TCP sender buffer according to $cwnd$ 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. Transmission control protocol variants

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 (AIMD) strategy, with various increase and decrease parameters. TCP NewReno [1] and Cubic [16] are examples of 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 [17] and Capacity and Congestion Probing (CCP) [3] are examples of delay based protocols.

Most TCP variants follow TCP Reno phase framework: slow start, congestion avoidance, fast retransmit and fast recovery. For TCP variants widely used today, congestion avoidance phase is sharply different. We will be introducing specific TCP variants' congestion avoidance phase shortly.

A. 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:

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

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). Parameters K drives the cubic increase away from Wmax, whereas β tunes how quickly cwnd reduction happens on packet loss. This process ensures that its $cwnd$ quickly recovers after a loss event.

B. 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 [17]. Compound TCP cwnd adjustment is as per (2):

$$\begin{aligned} AckRec: \quad cwnd_{k+1} &= cwnd_k + \frac{1}{cwnd_k + dwnd_k} \\ PktLoss: \quad cwnd_{k+1} &= \frac{cwnd_k}{2} \end{aligned} \quad (2)$$

where the delay component is computed as:

$$\begin{aligned} AckRec: dwnd_{k+1} &= \begin{cases} dwnd_k + \alpha dwnd_k^K - 1, & \text{if } diff < \gamma \\ dwnd_k - \eta diff, & \text{if } diff \geq \gamma \end{cases} \\ PktLoss: dwnd_{k+1} &= dwnd_k(1-\beta) - \frac{cwnd_k}{2} \end{aligned} \quad (3)$$

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.

C. 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 1 (b).

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) [15] and Opportunistic Linked Increase Algorithm (OLIA) [10]. 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.

D. Linked Increase Congestion Control

LIA [15] couples the congestion control algorithms of different sub-flows by linking their congestion window increasing functions, while adopting the standard halving of $cwnd$ window upon packet loss detection. More specifically, LIA cwnd adjustment scheme is as per (4):

$$\begin{aligned} AckRec: cwnd^i_{k+1} &= cwnd^i_k + min(\frac{\alpha B_{ack} Mss^i}{\sum_0^n cwnd^p}, \frac{B_{ack} Mss^i}{cwnd^i}) \\ PktLoss: cwnd^i_{k+1} &= \frac{cwnd^i_k}{2} \end{aligned} \quad (4)$$

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 (4) adopts $cwnd$ in bytes, rather than in packets (Maximum Segment Size - 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.

E. Opportunistic Linked Increase Congestion Control

OLIA [10] also couples the congestion control algorithms of different sub-flows, but with the increase based on the quality of paths. OLIA cwnd adjustment scheme is as per (5):

$$\begin{aligned} AckRec: cwnd^i_{k+1} &= cwnd^i_k + \frac{\frac{cwnd^i}{(RTT^i)^2}}{(\sum_0^n \frac{cwnd^p}{RTT^p})^2} + \frac{\alpha^i}{cwnd^i}, \\ PktLoss: cwnd^i_{k+1} &= \frac{cwnd^i_k}{2} \end{aligned} \quad (5)$$

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, 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.

V. Path Switching Aware MPTCP Packet Schedulers

MPTCP scheduler selects which sub-flow to inject packets into the network on a packet by packet basis. The default strategy is to select the path with shortest average packet delay. However, this greedy shortest delay strategy may increase the number of path switches on a streaming session. Lets first analyze the impact of path switching on application streaming performance.

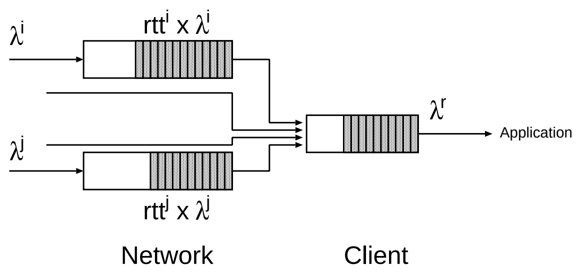


Figure 2: Path i to j Switching Model

Let λ^i, λ^j be the packet injection rates of a video stream session into path i and path j, respectively, as in Figure 2. Let also rtt^i, rtt^j be their respective round trip times. In addition, let λ^r be the playout buffer draining rate. Then:

- **Buffer underflow:** At the moment of path i to j switch, there are roughly $\lambda^i rtt^i$ packets in transit. As these packets get serviced at λ^r rate, for buffer underflow to occur, all these packets need to be serviced before the first packet injected at path j arrives. Assuming it takes rtt^j amount of time for this first packet to arrive, the condition for buffer underflow upon switching from path i to path j is: $rtt^i \lambda^i < rtt^j \lambda^r$. That is, buffer underflow probability is proportional to the ratio rtt^j/rtt^i. Hence, buffer underflow is more likely on path switches from smaller path rtt to larger path rtt.
- **Picture discard:** Assume F packets are needed to re-assemble a frame. Let F^i and $F - F^i$ be the number of packets transmitted on path i and path j, respectively. Then, in a transition from path i to path j, it takes F^i/λ^i to deliver these packets to playout buffer. By this time, the rest of the frame must have arrived at playout buffer, or some packets will be missing and the frame will not be able to be reassembled. It takes $(F - F^i)/\lambda^j$ to inject these packets, and another rtt^j delay for them to arrive at the playout buffer. So, the condition for frame discard is: $F^i/\lambda^i < (F - F^i)/\lambda^j + rtt^j$, or $F^i < (F - F^i)\lambda^i/\lambda^j + rtt^j \lambda^i$. That is, picture discard probability is proportional to the ratio λ^i/λ^j.

We study three path schedulers, seeking to minimize the number of path switches, as follows. On the onset of a video streaming session, the path with smallest rtt is chosen, as with the default path scheduler. However, once a new path is selected (due to congestion of previous path), three strategies for path switch minimization are studied: i) the scheduler stays on a new path for as long as it can, until the new path experiences congestion. We call this path scheduler greedy sticky scheduler - GR-STY; ii) the scheduler checks whether $\lambda^i/\lambda^j < 1$ before committing to stick to a new path. We call this version throughput sticky scheduler - TP-STY; iii) In addition to previous condition, the scheduler checks whether $rtt^i < rtt^j$. We call this version throughput RTT sticky scheduler - TR-STY. We evaluate these path schedulers against the minimum rtt default scheduler - DFT. Notice that our ultimate goal is to minimize buffer underflow and picture discards at the video receiver.

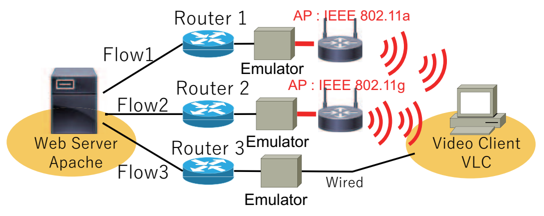


Figure 3: Video Streaming Emulation Network

TABLE I: EXPERIMENTAL NETWORK SETTINGS

Element	Value
Video size	409 MBytes
Video rate	5.24 Mbps
Playout time	10 mins 24 secs
Video Codec	H.264 MPEG-4 AVC
MPTCP variants	Cubic, Compound, LIA, OLIA
MPTCP schedulers	DFT, GR-STY, TP-STY, TR-STY

TABLE II: EXPERIMENTAL NETWORK SCENARIO

Scenario	Emulator configuration (RTT, Bandwidth, Random loss rate)
3 path Equal Loss Rate (Base Line Scenario)	Flow1) RTT 50 ms, BW 2 Mb/s, Loss 0 % Flow2) RTT 100 ms, BW 2 Mb/s, Loss 0 % Flow3) RTT 100 ms, BW 2 Mb/s, Loss 0 %
3 path Differential Loss Rate (3p-50)	Flow1) RTT 50 ms, BW 2 Mb/s, Loss 2.0 % Flow2) RTT 100 ms, BW 2 Mb/s, Loss 0 % Flow3) RTT 100 ms, BW 2 Mb/s, Loss 0 %
3 path Differential Loss Rate (3p-150)	Flow1) RTT 150 ms, BW 2 Mb/s, Loss 2.0 % Flow2) RTT 200 ms, BW 2 Mb/s, Loss 0 % Flow3) RTT 200 ms, BW 2 Mb/s, Loss 0 %

VI. Video Streaming Performance of Sticky Multipath Schedulers

In Figure 3, we describe the network testbed used for emulating a network path with wireless and wired access links. An HTTP Apache video server is connected to three access switches, which are connected to link emulators, 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), as well as a wired link. 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. This simple topology and isolated traffic allows us to better understand the impact of differential delays and packet loss on streaming performance.

We list network settings and scenarios generated by network emulator in Tables I and II, respectively. Video settings are typical of a video stream. Its size is short enough to enable multiple streaming trials within a reasonable amount of time. For each scenario, path bandwidth capacity is tuned so as to force the use of all three paths to stream a video playout rate of 5.24Mbps. We also inject 2.0 % of packet loss on the shortest path of each scenario except the baseline scenario, so as to contrast default packet scheduler (shortest RTT) with other schedulers. TCP variants used are: Compound, Cubic, LIA and OLIA.

Performance measures adopted 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.

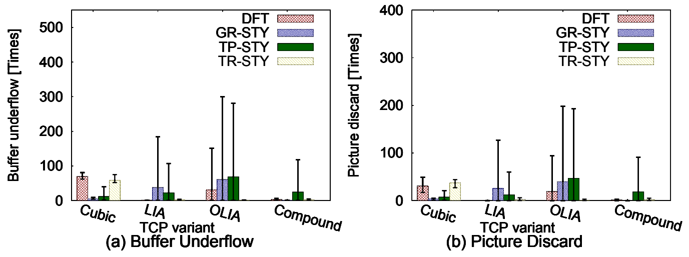


Figure 4: Scheduler Streaming Perf.; Base Line Scenario

- **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.
- **Sub-flow throughput:** the value of TCP Throughput on each sub-flow. This measure captures how MPTCP operates its scheduling packet injection and whether it is able to maintain a high enough throughput for the video playout rate.
- **Number of path switches:** number of path switches executed during a video streaming session. A path switch is counted every time two consecutive packets are injected into the network via two different subflows.

We organize our video streaming experimental results in three network scenarios: i) A baseline scenario, with no injected packet loss and differential delay; ii) Three path MPTCP under medium round trip delay; iii) Three path MPTCP under long round trip delay. Results are reported as average and min/max deviation bars.

A. Baseline Scenario

In Figures 4, a and b report on video streaming and TCP performance of baseline scenario, where 50, 100, and 100 msec delays are small, with only random packet loss from the wireless links. For Cubic variant, there is clearly a buffer underflow and picture discard performance improvement when GR-STY and TP-STY schedulers are used. TR-STY delivers similar performance to default scheduler, which can be explained by too low stickiness of TR-STY. On the other hand, LIA, OLIA and Compound TCP variants deliver best performance under default and TR-STY schedulers. We note that these three TCP variants are less aggressive in adjusting $cwnd$ than Cubic, as per respective equations of Section IV. Hence, it takes longer for these variants to grab newly available bandwidth of a new path.

In Figure 5, we report on the number of path switches executed for each scheduler, under various TCP variants. Firstly notice that GR-STY, TP-STY, and TR-STY schedulers have different levels of stickiness, GR-STY being the least restrictive. Hence, GR-STY delivers the lowest number of path switches, and TR-STY delivers the highest, closely following the number of path switches of the default scheduler. Notice also that the scheduler that delivers the lowest number of path switches is not necessarily the one that delivers best video performance in Figure 4. This shows the performance tradeoff between sticking to a given path, versus changing back to another path of perhaps better quality.

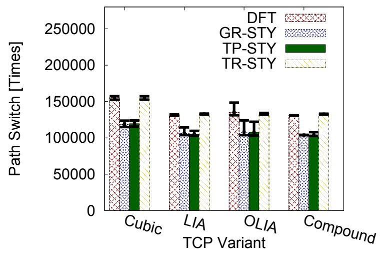


Figure 5: Path Switch.; Base Line Scenario

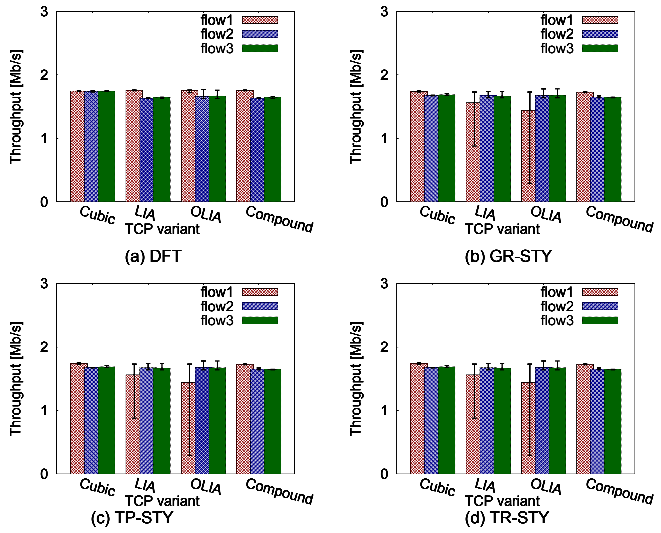


Figure 6: Throughput.; Base Line Scenario

In Figures 6 a, b, c, and d we report on throughput results of the various TCP variants operating under the several schedulers. For this analysis, it is important to call to attention that flow 3 is the best quality flow, as it is a wired path with no wireless random packet loss; flow 1 and flow 2 present random packet losses due to wireless link interference. So, from a throughput efficiency point of view, flow 3 should be preferred in detriment of the other ones. Notice that the default scheduler in Figure 6 a favors flow 1, due to its smaller rtt, whereas the other sticky schedulers divert more traffic away from flow 1 into flow 3, especially for slow to react LIA and OLIA TCP variants (in Figures 6 b, c, and d). This causes less overall retransmission (graphs omitted), resulting in a better transport efficiency.

B. Small delay with packet loss scenario

In Figures 7, a and b reports on video streaming performance under network scenario 3p-50, with short path delays of 50, 100, and 100 msecs, where 2.0 % random packet loss is injected into the shortest delay path. First notice how a relatively small packet loss rate causes significant buffer underflow and picture discard degradation as compared to the baseline scenario. About TCP variants, Cubic and Compound TCP improve their buffer underflow and picture discard performances under TR-STY schedulers, whereas LIA and OLIA variants present similar performance across all schedulers.

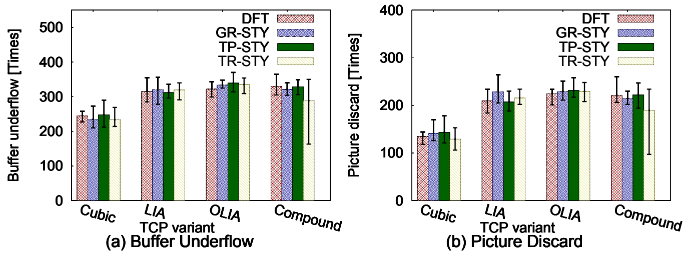


Figure 7: Scheduler Streaming Perf.; Scenario 3p-50

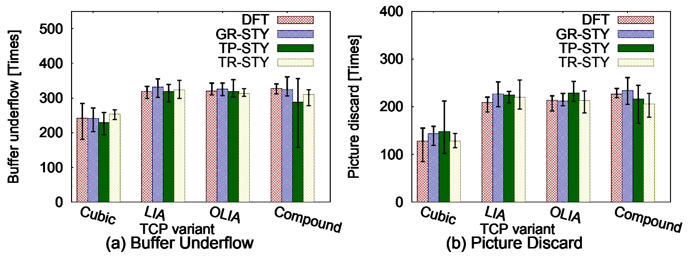


Figure 8: Scheduler Streaming Perf.; Scenario 3p-150

C. Large delay with packet loss scenario

In Figures 8, a and b reports on video streaming and TCP performance under scenario 3p-150, a three path scenario with large delays of 150, 200, and 200 msecs, respectively, with a 2.0 % random loss on shortest path. We can see that, when compared with previous scenarios, Cubic and Compound TCP variants have smaller buffer underflow and picture discard improvements between using different versions of Sticky scheduler and the default scheduler. LIA and OLIA present similar performance accross all schedulers. We conjecture that the larger the path delays are, the less performance improvement gains.

Overall, the above results show that video streaming performance improvement can be obtained by reducing path switching among available paths while avoiding path switches with high probability of causing buffer underflow and picture discards at the video receiver side. In addition, there seems to be a point of diminishing returns when paths have very long round trip times. Although these results were obtained for specific testbed topology and network scenarios, we believe similar improvements can be attained on more generic network scenarios.

VII. Conclusion and Future work

In this paper, we have proposed packet schedulers that reduce path switching among available paths to improve the quality of streaming video over MPTCP. We have evaluated MPTCP performance with default and packet schedulers which avoid path switching when the probability of buffer underflow and picture discard is high. Our results have shown that a clever path switch reduction may improve video streaming for Cubic and Compound Linux and Windows TCP variants, respectively, while not impacting performance of coupled LIA and OLIA variants. We believe that avoiding path switching may be applicable across a wide variety of network scenarios. We are currently investigating the integration of path switching management with other path scheduling mechanisms to improve video streaming performance.

Acknowledgments

Work supported by JSPS KAKENHI Grant # 16K00131.

References

[1] M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control,” IETF RFC 2581, April 1999.
[2] 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, pp. 42-48, September 2010.
[4] 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.
[5] E. Dong et. al., “LAMPS: A Loss Aware Scheduler for Multipath TCP over Highly Lossy Networks,” *Proceedings of the 42th IEEE Conference on Local Computer Networks*, pp. 1-9, October 2017.
[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] A. Frommgen, J. Heuschkel and B. Koldehofe, “Multipath TCP Scheduling for Thin Streams: Active Probing and One-way Delay-awareness,” IEEE Int. Conference on Communications (ICC), pp.1-7, May 2018.
[9] J. Hwang and J. Yoo, “Packet Scheduling for Multipath TCP,” IEEE 7th Int. Conference on Ubiquitous and Future Networks, pp.177-179, July 2015.
[10] R. Khalili, N. Gast, and J-Y Le Boudec, “MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution,” IEEE/ACM Trans. on Networking, Vol. 21, No. 5, pp. 1651-1665, Aug. 2013.
[11] Kimura et al., “Alternative Scheduling Decisions for Multipath TCP,” IEEE Communications Letters, Vol. 21, No. 11, pp. 2412-2415, Nov. 2017.
[12] Matsufuji et al., “Multipath TCP Packet Schedulers for Streaming Video,” IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) , August 2017, pp. 1-6.
[13] Nagayama et al., “TCP State Driven MPTCP Packet Scheduling for Streaming Video,” IARIA 10th International Conference on Evolving Internet, pp. 9-14, June 2018.
[14] 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.
[15] C. Raiciu, M. Handly, and D. Wischik, “Coupled Congestion Control for Multipath Transport Protocols,” IETF RFC 6356, 2011.
[16] I. Rhee, L. Xu, and S. Ha, “CUBIC for Fast Long-Distance Networks,” Internet Draft, draft-rhee-tcpm-ctcp-02, August 2008.
[17] 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.
[18] F. Silva, D. Bogusevschi, and G-M. Muntean, “A MPTCP-based RTT-aware Packet Delivery Prioritization Algorithm in AR/VR Scenarios,” In Proceedings of IEEE Intern. Wireless Communications & Mobile Computing Conference - IWCMCC 18, pp. 95-100, June 2018.
[19] 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.
[20] Xue et al., “DPSAF: Forward Prediction Based Dynamic Packet Scheduling and Adjusting With Feedback for Multipath TCP in Lossy Heterogeneous Networks,” IEEE/ACM Trans. on Vehicular Technology, Vol. 67, No. 2, pp. 1521-1534, Feb. 2018.
[21] F. Yan, P. Amer, and N. Ekiz, “A Scheduler for Multipath TCP,” In Proceedings of IEEE 22nd ICCCN, pp. 1-7, 2013.