FStream: Flexible Stream Scheduling and Prioritizing in Multipath-QUIC

Xiang Shi*, Lin Wang†‡, Fa Zhang* and Zhiyong Liu*
*Institute of Computing Technology, Chinese Academy of Sciences, China
†Vrije Universiteit Amsterdam, The Netherlands
‡Technische Universität Darmstadt, Germany
Email: shixiang@ict.ac.cn, lin.wang@vu.nl, zhangfa@ict.ac.cn, zyliu@ict.ac.cn

***Abstract*—While the web keeps evolving, web latency remains a major obstacle to improving user experience. In the past, many efforts have been made in this course. SPDY[1] achieves reduced latency through multiplexing and prioritization by manipulating HTTP. Quick UDP Internet Connection (QUIC) generalizes the idea and embeds multiplexing in the transport layer by introducing application-oriented streams. Multipath-QUIC brings further improvements by utilizing multiple paths as is done in MultiPath TCP (MPTCP). However, failing to account for stream priorities in the transport layer can result in suboptimal performance for time-critical streams.**

We fill this gap and propose FStream – a flexible stream scheduling mechanism for Multipath-QUIC, which provides stream prioritization down to the transport layer. We implement FStream in Multipath-QUIC and demonstrate its effectiveness in reducing the completion time of time-critical streams (~3x) through extensive experiments under different path dissimilarity conditions.

***Index Terms*—QUIC; MPQUIC; multipath; stream scheduling; prioritization; web latency**

I. Introduction

With the increase in the richness of modern web content, the number of web domains and objects continues to grow fast. However, web transfer latency still remains as the major impediment to improving user-perceived performance. Over the years, many approaches have been proposed. In HTTP/1.0, a new TCP connection is initiated for every single request, incurring a dramatic overhead to the overall completion time due to the slow start of TCP connections. HTTP/1.1 improves this situation by enabling request pipelining. However, the request concurrency still suffers from head-of-line (HOL) blocking problem due to restrictions on the returning order of responses – a single slow response can block all responses behind it. To further mitigate HOL blocking, SPDY [2] multiplexes HTTP requests over a single TCP connection. However, since the underlying TCP connection of SPDY requires in-order transmission, HOL blocking still exists in the transport layer. As an important goal, Quick UDP Internet Connection (QUIC) proposes to bring multiplexing down to the transport layer [3]. QUIC multiplexes HTTP requests/responses over UDP by providing each with an application-oriented stream (stream for short).

Besides HOL blocking, another important issue exists, which was recognized by SPDY: If the bandwidth of a channel is limited, the client may block requests to avoid overwhelming the channel. It may happen that a time-critical request is blocked by non-critical ones. To overcome this problem, SPDY adds priorities to requests, where the client can send unlimited requests with each assigned a priority. Unfortunately, many QUIC implementations have not supported such functionality at this moment and failing to account for priorities of streams can result in suboptimal performance, especially for time-critical streams.

To enable handover and aggregate the bandwidth of different paths with QUIC, Multipath-QUIC (MPQUIC) was proposed [4]. With the stream-multiplexed feature of QUIC, fine-grained scheduling on multipath can be achieved considering the different preferences of different streams. In a default setting, MPQUIC always schedules packets on the path with the lowest round-trip time (RTT). However, HTTP/2 prioritization is not incorporated into the scheduler since the streams are treated without differences. As a result of resource contention, small streams that are responsible for time-critical requests can be easily blocked by non-critical requests, especially those long-lived large streams. SA-ECF [5] makes per-packet scheduling decisions based on stream completion time estimation and selects from the two paths with the smallest RTT. However, streams are with different resource preferences for various application demands. To illustrate, throughput-sensitive streams prefer high bandwidth paths to low RTT paths. Current schedulers do not take the resource preferences of streams into consideration. Therefore, we argue that treating all the streams undifferentiated can be suboptimal, and thus affecting the stream completion time.

In this paper, we propose a flexible stream scheduler called FStream for MPQUIC. Upon a stream arrival, we schedule the stream to its suitable path. FStream is designed with three main features: 1) Packets are scheduled on a per-stream granularity, preventing out-of-order and aggregation delay brought by multiple paths. 2) Time-critical streams

X. Shi is also with University of Chinese Academy of Sciences, Beijing, China. The Corresponding Author is Zhiyong Liu (zyliu@ict.ac.cn). This work was partially supported by the National Key Research and Development Program of China (grant number 2017YFB1010001), the National Natural Science Foundation of China (grant numbers 61520106005, 61761136014). L. Wang was funded by the German Research Foundation (DFG) project 392046569 and by the subproject C7 within the DFG Collaborative Research Center 1053 (MAKI).

[1]SPDY has been subsumed by the HTTP/2 standard [1].

978-1-7281-2583-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPADS47876.2019.00136

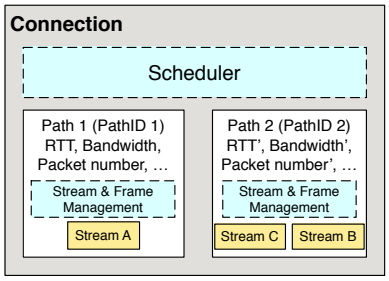


Fig. 1. An overview of an FStream connection. The blocks with dash-line borders show our modifications.

are prioritized in scheduling and are more likely assigned to paths with favorable resources. 3) Streams scheduled to the same path share bandwidth proportionally to their priorities, preventing the case that streams that are more time-critical are blocked by the others because of resource contention.

II. FStream Design and Implementation

In the application layer, a client (i.e. browser) sends requests to the webserver with each assigned an HTTP/2 priority (priority for short) [6] between 1 and 255 (inclusive). We use this value as an indication of the urgency of the corresponding initiated streams in FStream. Higher priority value means the client expects it to complete earlier.

Fig. 1 illustrates the main structure of an FStream connection. A connection is established after the cryptographic handshake of MPQUIC and is identified by a Connection ID (CID). There can be multiple available paths within a connection. Each path is scheduled from none to multiple streams and has a stream manager that manages the transmission of the streams. A global scheduler is responsible for the stream-to-path scheduling based on our proposed scheduling policy that will be detailed in the following.

Scheduler design. The scheduler is designed to schedule streams to appropriate paths. Note that as the stream dependencies restrict the initiation order of streams, we consider the path scheduling of concurrently initiated streams each time. In order to come up with an effective scheduler, we observe the following principles to be followed by the scheduling policy.

- **PRCP-1** Streams with higher priorities should be prioritized in the order of being scheduled.
- **PRCP-2** Packets of each stream should be scheduled to follow a single path, preventing stream completion from being blocked by slow paths.
- **PRCP-3** Higher priority streams should be prioritized over lower priority streams on the same path.

For streams each responsible for an HTTP request, their sizes are usually known at the server-side before transmission. Therefore, we can schedule them to appropriate paths following these principles.

To satisfy PRCP-1, the scheduler sorts the streams in descending order of priority upon arrival of concurrent streams. According to the priority order, the scheduler performs the following scheduling procedure one by one. For a stream s_i, the scheduler calculates a rough estimate of transmission time T for the stream on each path p_j using the following equation:

$$T(s_i, p_j) = \frac{SIZE_i}{b_{ij}} + \frac{RTT_j}{2} \tag{1}$$

where $SIZE_i$ is the object size to be transmitted on stream s_i, and RTT_j is the estimated round-trip time of the path p_j. b_{ij} is the bandwidth share stream s_i can receive if it is scheduled to path p_j following the proportional sharing principle. We suppose the sum of priority of all the streams that are already scheduled onto path p_j is SUM_j, and the priority value of stream s_i is PRT_i, and then b_{ij} can be calculated by:

$$b_{ij} = \frac{PRT_i}{SUM_j + PRT_i} \cdot BW_j \tag{2}$$

where BW_j is the smoothed bandwidth estimation of path p_j. This bandwidth-sharing mechanism is designed based on PRCP-3 to provide prioritization for streams sharing the same path. After finishing the transmission time estimation, we choose the suitable path for each stream with the minimum estimation $\min T(s_i, \cdot)$, which respects PRCP-2.

Implementation. To validate the effectiveness of our proposal, we complete a proof-of-concept implementation based on existing MPQUIC open-source implementation written in Go [7]. We divide the available bandwidth of the path in proportion to the stream priorities. For streams on the same path, we achieve the bandwidth-sharing mechanism of the streams in a probabilistic way. When a path sends data in one transmission round, we choose a stream to transmit and we calculate the probability of each stream to be transmitted on the path. We first obtain the priority sum by adding up the priority values of all the normal streams on the path. Then, we divide the priority of the stream by the priority sum to calculate the probability of the stream to be selected. Arriving streams will be scheduled to appropriate paths based on their priorities and sizes. As a starting point, we set up initial values of RTT and bandwidth based on prior knowledge on creating a path; other techniques can also be used [8], [9]. During the transmission, we maintain smoothed RTT estimation values of each path. The RTT estimation is based on [8] and the bandwidth estimation is based on [10].

III. Experiments

We evaluate our mechanism FStream with comparison to the original MPQUIC implementation and SA-ECF [5] on the Mininet emulation platform [11]. To provide a fair assessment of the compared implementations, we adopt an experimental design similar to the one used for MPQUIC [4]. We evaluate the completion time of the concurrently transmitted time-critical stream and non-critical stream under a wide range of parameters for available paths.

A. Setup

Network. We consider the same network topology with two multi-homed hosts over disjoint paths used by [4] [5].

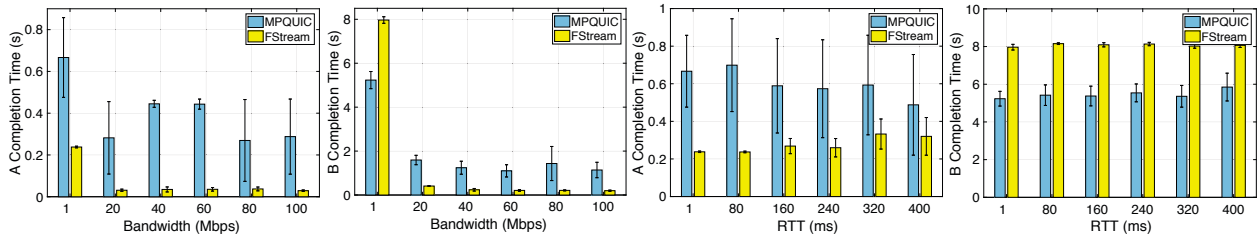


Fig. 2. The impact of path bandwidth and RTT dissimilarity on the completion time of stream A and B respectively: (a) and (b) bandwidth dissimilarity, (c) and (d) RTT dissimilarity. Each repeated simulation carries a standard deviation value.

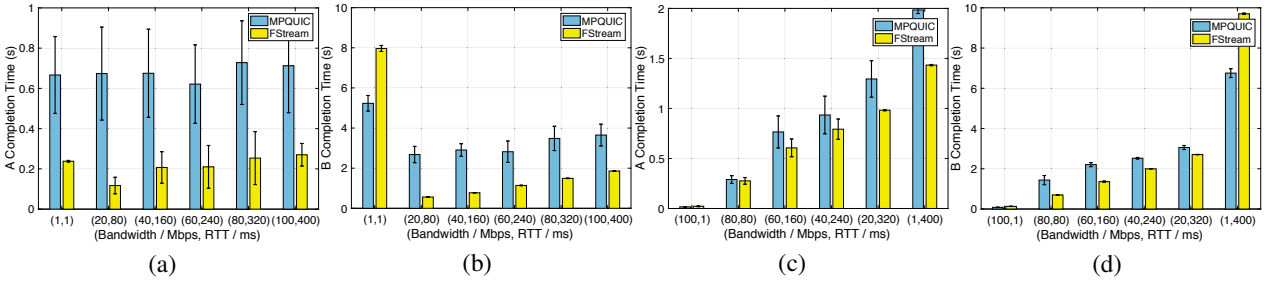


Fig. 3. The impact of changing both bandwidth and RTT of path #2 at the same time: (a) and (b) each path with its advantage, (c) and (d) one path wins over the other. Each repeated simulation carries a standard deviation value.

This topology allows us to evaluate multipath-scenarios like a mobile phone connecting to both Mobile broadband (MBB) and WIFI at the same time [5]. To evaluate the impact of path dissimilarity on the performance, we set different kinds of dissimilarities for the two paths. For each path, we focus mainly on the bandwidth and RTT of the path. Note that packet losses caused by router buffer overflow can occur in these environments. We set the range of bandwidth from 1Mbps to 100Mbps and RTT from 1ms to 400ms [4]. The maximal receive window is set to 16 MB. We perform our evaluation on a laptop with an Intel Core i5 Dual-Core CPU i5-4278U@2.60GHz with 8.0GB RAM.

Bandwidth dissimilarity. We set the bandwidth of path #1 to be 1Mbps and RTT of both paths to be 1ms. We vary the bandwidth of path #2 from 1Mbps to 100Mbps to increase the bandwidth dissimilarity of the two paths.

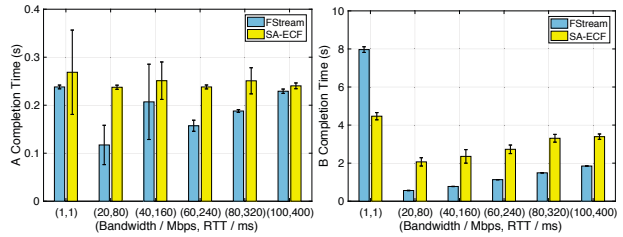
RTT dissimilarity. We set the bandwidth of both paths to be 1Mbps and RTT of path #1 to be 1ms. Then we vary the RTT of path #2 from 1ms to 400ms to increase RTT dissimilarity.

Bandwidth and RTT dissimilarities. We set up equivalent initial values of path #1 and path #2, i.e., bandwidth to be 1Mbps and RTT to be 1ms. Firstly, we increase the bandwidth and RTT values of path #2 at the same time. Secondly, path #1 and path #2 are set initially with bandwidth to be 1Mbps and RTT to be 400ms. Then we increase the bandwidth and decrease the RTT values of path #2 simultaneously.

Traffic. We evaluate the completion time of two concurrently initiated streams corresponding to two types of web objects inside a web page. This prioritization strategy (i.e. priority assignment and the order of initiating streams) is based on the weighted round-robin mechanism of Safari 11 [6]. The first is a time-critical small stream A corresponding to an HTML document, with 26 KB size and priority value to be 255. The second is a delay-tolerant big stream B corresponding to an image, with 900KB size and priority value to be 8. The client measures the completion time of a stream by recording the time elapsed between the initiation of the request and the reception of the last byte of the corresponding stream. Each simulation is repeated 6 times for all the schedulers, and we analyze the average values.

B. Experimental Results

Fig. 2(a)(c) and Fig. 3(a)(c) illustrate the completion time of the time-critical small stream A. We can observe that FStream performs better than the original mechanism of MPQUIC in general. *With prioritization enabled, FStream is especially beneficial for reducing the completion time of time-critical transmissions.* The reason is that in FStream, stream A is prioritized in resource sharing and scheduling order ensuring that it will not be blocked by the non-critical stream. Meanwhile, packets of stream A are scheduled to follow a single path, thus preventing aggregation delay brought by multiple paths. In MPQUIC, packets of stream A are always transmitted on the path with available window and lower RTT, thus the packets can be distributed over the two paths. As a result, the completion time can suffer from significant fluctuation in the repeated simulations. In Fig. 3(c), the performance differences are less evident than the other figures of stream A, which is due to the overall advantage of path #2 over path #1. In this case, MPQUIC chooses mostly path #2 for transmission, while FStream schedules stream A on path #2. When the advantage of path #2 is getting weaker along the x-axis of



(a) (b)

Fig. 4. Performance results of FStream and SA-ECF with path dissimilarities of both bandwidth and RTT : (a) the completion time of stream A, and (b) the completion time of stream B.

Fig. 3(c), MPQUIC chooses path #2 at a reduced chance for each packet, and FStream still schedules stream A on path #2 for the advantage, thus the performance differences are becoming more evident.

Fig. 2(b)(d) and Fig. 3(b)(d) depicts the completion time of the big stream B with low priority. We can see that in most cases, the completion time of stream B is reduced as well as stream A. However, there are cases that FStream performs not as good as MPQUIC for stream B. This can happen in the situations that the bandwidths of the two paths are homogeneous (i.e. Fig. 2(d), the first bar in Fig. 2(b) and Fig. 3(b), and the last bar in Fig. 3(d)). The reason is that due to the mechanism of FStream, stream B is scheduled to the other path and cannot utilize the path occupied by A to avoid bandwidth contention. This is due to the considerations that the design rationale of FStream is to sacrifice the completion time of a non-critical stream as a tradeoff to avoid resource contention between streams that can affect the completion time of a time-critical stream.

C. Comparison to SA-ECF

Following the same experiment settings, we also conduct experiments to compare the performance differences between FStream and SA-ECF. SA-ECF [5] provides a packet scheduler for MPQUIC to avoid transmission delayed by slower paths. For each packet, SA-ECF first chooses two paths: One path is with the lowest RTT and not necessarily to be an available path, while the other is an available path with the lowest RTT. Then SA-ECF estimates the completion time of the stream to which the packet belongs on two paths, and schedules the packet to the path with the lowest estimated time. Fig. 4 shows the results of FStream and SA-ECF. It is clearly shown that the overall performance of FStream is better than those of SA-ECF. In the cases with path heterogeneity, FStream reduces the completion time of stream A as well as stream B, for each stream is scheduled to its favorable path. While in SA-ECF, packets of two streams can share the bandwidth of the same path, leading to bandwidth contention between stream A and B. Meanwhile, the packets of stream B can be transmitted on path #1 with the lower bandwidth, leading to a longer completion time. In the case that paths are homogeneous (i.e. the first bars in Fig. 4), FStream reduces the completion time of stream A, but it deteriorates the completion time of stream B. This is because FStream sacrifices the bandwidth aggregation of a non-critical stream to trade for a performance gain for a time-critical stream.

IV. Conclusion and Future Work

In this paper, we proposed a flexible stream scheduling mechanism, FStream, for MPQUIC. FStream provides stream prioritization at the transport layer, preventing streams that are responsible for time-critical requests from being blocked by non-critical requests. A proof-of-concept implementation and extensive experiments proved the effectiveness of FStream, showing that it outperforms the original scheduling mechanism of MPQUIC in reducing the completion time of time-critical streams in different path heterogeneity. As to future work, we are interested in extending FStream to provide improvements for throughput-sensitive streams, and we will explore the performance of FStream when being coupled with different prioritization strategies of different browsers.

References

[1] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext transfer protocol version 2 (HTTP/2). *RFC*, 7540:1–96, 2015.

[2] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. How speedy is spdy? In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 387–399, 2014.

[3] Adam Langley et al. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 183–196, 2017.

[4] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: design and evaluation. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2017, Incheon, Republic of Korea, December 12 - 15, 2017*, pages 160–166, 2017.

[5] Alexander Rabitsch, Per Hurtig, and Anna Brunström. A stream-aware multipath QUIC scheduler for heterogeneous paths: Paper # xxx, XXX pages. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ@CoNEXT 2018, Heraklion, Greece, December 4, 2018*, pages 29–35, 2018.

[6] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. HTTP/2 prioritization and its impact on web performance. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1755–1764, 2018.

[7] De Coninck Q et al. https://github.com/qdeconinck/mp-quic.

[8] Van Jacobson. Congestion avoidance and control. In *SIGCOMM '88, Proceedings of the ACM Symposium on Communications Architectures and Protocols, Stanford, CA, USA, August 16-18, 1988*, pages 314–329, 1988.

[9] Claudio Casetti, Mario Gerla, Saverio Mascolo, M. Y. Sanadidi, and Ren Wang. TCP westwood: End-to-end congestion control for wired/wireless networks. *Wireless Networks*, 8(5):467–479, 2002.

[10] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: congestion-based congestion control. *ACM Queue*, 14(5):20–53, 2016.

[11] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Conference on emerging Networking Experiments and Technologies, CoNEXT '12, Nice, France - December 10 - 13, 2012*, pages 253–264, 2012.