PriorityBucket: A Multipath-QUIC Scheduler on Accelerating First Rendering Time in Page Loading

Xiang Shi[‡]
shixiang@ict.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China

Fa Zhang[†]
zhangfa@ict.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China

Zhiyong Liu[*]
zyliu@ict.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China

ABSTRACT

Page load time is a web performance metric that directly impacts user experience. It measures the time it takes to fully load a web page after making a request. However, as the first visual feedback of a web page, first rendering time is also a key metric to satisfy the users. In this paper, we focus on speeding up the first rendering time in page loading from the perspective of protocol improvements. We observe that by prioritizing the streams that make up the critical rendering path in Multipath-QUIC (MPQUIC), the first rendering time can be effectively reduced. Therefore, we propose Priority-Bucket – an MPQUIC scheduler that provides stream prioritization in the transport layer. By comparing to the existing schedulers of MPQUIC, we show its effectiveness in reducing the first rendering time and page load time of web pages in different path heterogeneity using the Chrome browser. Results show that PriorityBucket can reduce the first rendering time and page load time up to 34% and 35% respectively in high path heterogeneity when loading Bing.

CCS CONCEPTS

• **Networks → Transport protocols**; **Network protocol design**.

KEYWORDS

QUIC, MPQUIC, first rendering time, stream scheduling

ACM Reference Format:
Xiang Shi, Fa Zhang, and Zhiyong Liu. 2020. PriorityBucket: A Multipath-QUIC Scheduler on Accelerating First Rendering Time in Page Loading. In *The Eleventh ACM International Conference on Future Energy Systems (e-Energy'20), June 22–26, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3396851.3402923

1 INTRODUCTION

Web browser has become one of the most commonly used applications for desktop and mobile users. With the increase of web page complexity in recent years, page load time has become the critical metric to impact the quality of user experience. Page load time measures the time it takes for a page to fully load in a browser after a user clicks a link or makes a request.

To improve user-perceived performance, many approaches have been proposed to reduce page load time. In the application layer, SPDY [1] multiplexes HTTP requests over a TCP connection to solve the head-of-line blocking (HOL) problem due to restrictions on the returning order of responses. However, since the underlying TCP connection of SPDY requires in-order transmission, HOL still exists in the transport layer. As an important goal, Quick UDP Internet Connection (QUIC) proposes to bring multiplexing down to the transport layer [2]. QUIC multiplexes HTTP requests/responses over a single UDP connection by providing each with a stream. With UDP as its substrate, QUIC guarantees that a late packet for each individual stream will not block other concurrent streams. To further reduce web latency, Multipath-QUIC (MPQUIC) [7] was proposed by enabling handover and aggregating the bandwidth of different paths with QUIC.

However, a web browser may begin rendering a web page before all resources have been fetched. The purpose is to give visual feedback to users and to make the page interactive as soon as possible. The period from a blank page to its first visual feedback is called the first rendering time [3], which is also a critical metric in the page loading process. Short first rendering time shows the responsiveness of the web server, thus can prevent the user from closing the tab.

The first rendering time is decided by the critical rendering path, which is a series of steps that web browsers need to perform before they start rendering web pages [3]. These steps require the browser to evaluate the HTML, CSS, and JavaScript that make up the web page. First, the browser builds the document object model (DOM) tree from HTML and CSS Object Model (CSSOM) tree from any potential CSS file. Then, it combines the DOM tree and CSSOM tree into the final rendering tree used to render the page. In this process, some JavaScript can potentially modify DOM and CSSOM trees, so it may also be necessary to render the page correctly. Other types of web objects (except HTML, CSS, and JavaScript) are not required in the critical rendering path. Therefore, by giving priority to the streams that make up the critical rendering path (referred to as the critical streams), first rendering time can be reduced.

HTTP/2 [4] performs prioritization on HTTP requests, which can be represented as a dependency tree. The dependency tree is constructed with streams as nodes. In the tree, each stream is assigned a priority value, indicating the proportion of resources in concurrent streams. Meanwhile, a dependency relationship defines the transmission order of the streams. A stream that depends on another stream should not be allocated resources until the dependent stream has finished transmission. A recent study [5] investigates the impact of HTTP/2 prioritization and confirms that prioritization improves web performance after careful alignment with other components of the webpage load process.

In the transport layer, content prioritization can be achieved with finer granularity in QUIC and its variant as stream-multiplexed protocols. We observe this and propose a multipath scheduler for MPQUIC, which provides stream prioritization aiming at speeding first rendering time in loading web pages. We implement the scheduler in the MPQUIC go-implementation project [6], and then evaluate its performance over a range of network scenarios in a virtualized network setup. We compare it to other schedulers of MPQUIC during the evaluation.

The rest of the paper is organized as follows: Section II introduces the related work and Section III describes the design rationale of PriorityBucket in detail. Section IV discusses its implementation and Section V analyzes the experimental results. Section VI concludes the paper and discusses future work.

2 RELATED WORK

Multipath-QUIC (MPQUIC) [6] [7], as a important variant of QUIC, was proposed aiming at enabling handover and aggregating the bandwidth of multiple paths. The performance of a multipath protocol can be highly affected by the mechanism of scheduling packets over multiple paths. As a result, works have been done to investigate into the packet scheduling of MPQUIC.

The prototype of MPQUIC [6] disregards stream priority and packs a packet using a round-robin mechanism that cycles through the streams. For each packed packet, MPQUIC adopts the Lowest-RTT-First (LRF) scheduler. The scheduler uses round-trip time (RTT) as the only criterion for path scheduling. However, HTTP/2 prioritization is not incorporated into the prototype [6] and failing to account for priorities of streams can result in suboptimal performance [8], especially for time-critical streams. In addition, other path characteristics like bandwidth should also be considered in scheduling when facing path heterogeneity [9].

SA-ECF [10] incorporates HTTP/2 prioritization and makes per-packet scheduling decisions using both RTT and congestion window statistics of a path. It selects two paths with the lowest RTT and chooses a better one from the two paths. The decision is based on the completion time estimation of the remaining bytes of the stream. This is to ensure that the transmission is not delayed by sending over the slower path. However, concurrent streams with various priorities share the bandwidth of one path. Consequently, the critical streams can be delayed due to bandwidth contention between the streams. Moreover, the estimation uses the remaining bytes instead of the size of the stream. As a result, packets of different streams compete for the fast path in a greedy fashion without the full picture of the stream size, causing burst transmission [9] on the fast path.

Mogensen et al. [11] consider reliability as the major design goal and present a scheduler for MPQUIC where data is scheduled redundantly over multiple paths for real-time vehicular communication in LTE networks, while our major goal is to reduce web latency by prioritized scheduling. [12] assumes the dependency tree of a web page is known in advance at the server-side and then uses this information to optimize the transmission order of all the streams, which is not practical for online cases. [13] considers both stream size and priority features in scheduling. To further optimize the completion time of the time-critical stream, it schedules a stream to a single path. This sacrifices the bandwidth aggregation benefits of multipath transmission.

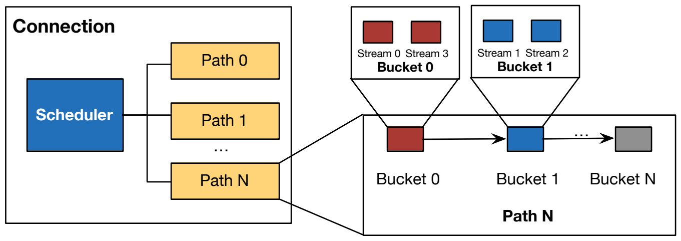

Figure 1: A structure of PriorityBucket design.

3 PRIORITYBUCKET DESIGN

To accelarate the first rendering time in page loading, PriorityBucket is proposed as an MPQUIC scheduler by optimizing the completion time of the critical streams. Since the HTTP/2 prioritization [5] indicates the criticality of a stream, we incorporate our scheduler with HTTP/2 to obtain the priority and size features of a stream.

3.1 Overview

After the connection is established, each arrival of a stream triggers the scheduler. Based on the features of the stream, the scheduler allocates the stream to paths with the calculated amount of data. When the allocation is completed, the transmission will be executed. This process ends when all streams finish transmitting and no stream arrives.

3.2 Structure

Fig. 1 illustrates the main structure of PriorityBucket design. A connection is established after the cryptographic handshake of MPQUIC and is identified by a Connection ID (CID). Within a connection, there can be multiple paths. Each path is scheduled from none to multiple streams, which are classified into buckets by priority. A global scheduler is responsible for the stream-to-path scheduling based on our proposed scheduling policy.

3.3 Stream and Bucket

A stream is a lightweight bi-directional byte stream abstraction that has a unique stream ID. Streams can be opened freely after the QUIC connection establishment, and can be cancelled or closed without tearing down the entire connection. A stream can be scheduled to one or multiple paths based on the calculated data portion by the scheduler.

To prioritize the critical streams on a path, we distinguish streams with different priority by buckets. Streams of equal priority value

Table 1: Scheduling Parameters

Symbol	Meaning
D	total data volume of stream
h_i	total data volume of streams with higher priority
e_i	total data volume of streams with equal priority
T	overall completion time of stream
i	path i
n	number of paths
o_i	estimated one-way delay of the path i
b_i	estimated bandwidth of the path i
d_i	fraction of data assigned to path i
t_i	completion time of stream s on path i

are divided into the same bucket. Inside the same bucket, the transmission order of streams follows the exclusive and first come first served (FCFS) nature. A bucket with lower priority depends exclusively on the most recent pending stream in a higher priority bucket. This results in a dynamic FCFS variant, where the transmission of in-flight lower-priority buckets can be pre-empted by critical streams in higher-priority buckets.

3.4 Scheduler

The scheduler decides the paths for a stream to perform transmission. Concurrently initiated streams are handled in descending order of priority. The resource preference of a stream is closely related to its size feature. For instance, big streams prefer paths with higher bandwidth, and small streams favor paths with lower RTT. Therefore, we consider stream size in our scheduling and assign a stream in a calculated proportion to each path after careful assessment.

3.4.1 Problem Description. The scheduling of a stream can be described as an optimization problem. In the scheduling of a stream, the scheduler needs to decide the amount of data to be allocated to each path. Since the overall completion time of a stream can be delayed by the slowest path, the transmission time on each path should be balanced for the stream. Assume there are n paths, we define T for the overall completion time of the stream s and t_i for the estimated completion time on each path. The data volume D of the stream s is split into portions and d_i is the data amount for the path i. Let h_i be the total size of the higher priority streams on path i, and e_i be the sum size of the equal priority streams on path i, then t_i can be calculated by

$$t_i = \begin{cases} 0, & d_i = 0 \\ \dfrac{h_i + e_i + d_i}{b_i} + o_i, & d_i > 0 \end{cases} \tag{1}$$

where o_i is the estimated one-way delay of the path i and b_i is the bandwidth estimation of path i. Then the scheduling problem can be summarized and formulated as follows:

$$\begin{aligned} &(P_1) \quad \min T \\ &\text{subject to} \\ &t_i = \begin{cases} 0, & d_i = 0 \\ \dfrac{h_i + e_i + d_i}{b_i} + o_i, & d_i > 0 \end{cases} \\ &\textstyle\sum_{i=1}^{n} d_i = D \\ &T = \max\{t_i \mid i \in [1, 2, ..., n]\} \end{aligned} \tag{2}$$

Algorithm 1 PriorityBucket algorithm

```
1: procedure ScheduleStream
2:     $sortedList \leftarrow$ sort paths by ascending order of $r_i$
3:     initialize $d_i = 0, i \in [1, 2, ..., n]$
4:     $D' \leftarrow D$
5:     for path $i$ in $sortedList[1, ..., n-1]$ do
6:         $r_{gap} \leftarrow r_{i+1} - r_i$
7:         if $r_{gap} \neq 0$ and $D' > 0$ then
8:             if $D' >= r_{gap} \times \sum_{k=1}^{i} b_k$ then
9:                 for $j \leftarrow 1$ to $i$ do
10:                    $inc \leftarrow r_{gap} \times b_j$
11:                    $d_j \leftarrow d_j + inc$
12:                    $D' \leftarrow D' - inc$
13:                end for
14:            else
15:                for $j \leftarrow 1$ to $i$ do
16:                    $inc \leftarrow D' \times (b_j / \sum_{k=1}^{i} b_k)$
17:                    $d_j \leftarrow d_j + inc$
18:                end for
19:                $D' \leftarrow 0$
20:                break
21:            end if
22:        end if
23:    end for
24:    if $D' > 0$ then
25:        for path $i$ in $sortedList$ do
26:            $d_i \leftarrow d_i + D' \times (b_i / \sum_{k=1}^{n} b_k)$
27:        end for
28:    end if
29: end procedure
```

3.4.2 Solution. To solve the model, we propose an algorithm and the pseudo-code is illustrated in Algo.1, which is composed of a procedure of three steps.

Step 1: If we schedule stream s to path i, then the minimum of completion time on path i is determined by $r_i = \frac{h_i + e_i}{b_i} + o_i$. Therefore, we sort paths by the ascending order of r_i.

Step 2: The *fillgap* process schedules data to balance the completion time gap of multiple paths. We start the process from $path_1$ with the smallest r_1, and compare r_1 with the second smallest r_2. We schedule data of $(r_2 - r_1) \times b_1$ on $path_1$ to make the completion time on $path_1$ to equal r_2. Then we compare r_2 and r_3. We fill the gap through scheduling data of $(r_3 - r_2) \times b_1$ on $path_1$ and data of $(r_3 - r_2) \times b_2$ on $path_2$. The process will be repeated until all the data is allocated or the completion time of all the paths becomes equivalent.

Step 3: Third, the remaining data are allocated proportionally to the bandwidth of paths to make the completion time balanced on multiple paths.

The scheduler performs the algorithm for each arrived stream, and assigns the calculated amount of data to the paths. This number serves as the upper sending limit of each stream on a path. The time complexity of the algorithm is $O(n^2)$. However, since each stream only needs to be calculated once, the algorithm did not bring noticeable overhead in our observation of simulation. Meanwhile, due to the limited number of paths in the actual network, we believe this also applies to real cases.

4 IMPLEMENTATION

We complete a proof-of-concept implementation based on existing MPQUIC open-source implementation written in Go [6]. The details of implementation are elaborated in the following parts.

4.1 Scheduler

The scheduler is triggered by stream arrival, and then schedules the paths for the stream. The scheduling requires bandwidth and RTT estimation of all the paths. 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 [14], [15]. During the transmission, we maintain smoothed values of RTT estimation of each path. The RTT estimation is based on the exponential weighted moving average mechanism of TCP [14]. And the bandwidth estimation is based on [16]. After the scheduling, the stream is scheduled to paths with a calculated amount of data as the sending limit. The scheduling performs once for each stream and we acknowledge that periodic updating of the scheduling results can be more beneficial under dynamic network conditions. However, frequent updating adds overhead to the system. Therefore, the periodic parameter needs to be carefully tuned and we left this as a future work.

4.2 Transmission

The transmission is performed in a round robin fashion with paths. Each path has a stream manager that manages the transmission of the scheduled streams. Streams on a path are divided into buckets by priority. The path determines the transmission order based on the priority of the bucket. Low priority buckets can only be transmitted after high priority buckets complete the transmission, and can be preempted by newly arrival at high priority buckets. Inside a bucket, the path transmits the earliest arrived stream until its completion or sending limit. Then the path continues to the next stream in the same bucket, or moves to the next bucket if there are no stream to be transmitted in this bucket.

5 EXPERIMENTS

We evaluate our mechanism PriorityBucket with comparison to the LRF scheduler [6] and the SA-ECF scheduler [10] on the Mininet emulation platform [17].

5.1 Setup

Network. To provide a fair assessment of the compared implementations, we adopt an experimental design similar to the one used for

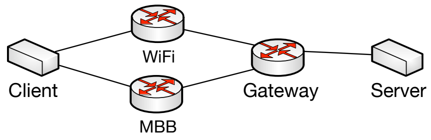

Figure 2: A typical multipath scenario over two network paths, which we used for our evaluation.

MPQUIC [6]. We consider the typical network topology with two multi-homed hosts over disjoint paths used by [6][10], as depicted in Fig. 2. This topology allows us to evaluate multipath-scenarios like a mobile phone connecting to both MBB and WiFi at the same time [10]. To evaluate the impact of path parameters on the performance, we evaluate the schedulers under different heterogeneity and packet loss rate (PLR), as listed in Table 2. The aggregated bandwidth remains at 50 Mbps. 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.
Traffic. We evaluate the performance on two diverse types of web pages using the Chrome browser [5]. The web pages are from the dataset [18]. As one of the most popular search sites, Bing is a simple type of web page that contains only two web objects, an HTML and an image. For complex web pages, the page load time is longer, so it is important to attract more users by keeping short first rendering time. We choose the YouTube website to represent the complex web pages, for its richness in types and quantity of web objects. The YouTube website includes one HTML, two CSS, two JavaScript, and twenty image objects. Each simulation is repeated ten times for all the schedulers, and we analyze the average values.
Metric. We evaluate the first rendering time and page load time of the web pages. We only focus on the network transmission time of the web objects, ignoring the processing time in the browser. The first rendering time means the loading speed of the first non-blank page, which is determined by the last finished critical stream. The page load time is measured by recording the time elapsed between the initiation of the request and the receipt of all the web objects in the web page.

5.2 Compared Schedulers

Lowest-RTT-First (LRF). LRF [6] round-robinly packs data of all the streams for a single packet. Upon sending the packet, an LRF sender needs to select a path for it. It maintains a smoothed round-trip time (RTT) estimation of each path. Each time sending a packet, it prefers the lowest RTT path with available congestion window.
SA-ECF. SA-ECF [10] provides a packet scheduler 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. The number of opportunities for each stream to send is proportional to its weight.

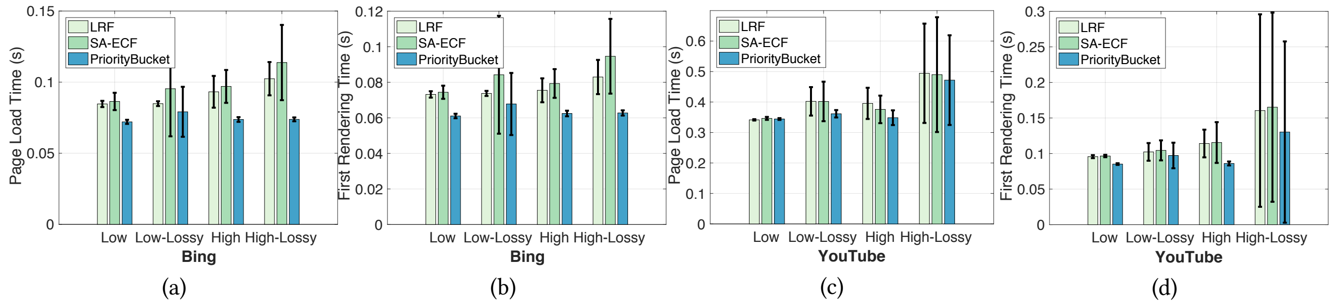

Figure 3: The performance of different schedulers on Bing and YouTube under various network settings: (a) is the page load time of Bing; (b) is the first rendering time of Bing; (c) is the page load time of YouTube; and (d) is the first rendering time of YouTube. Each repeated simulation carries a standard deviation value.

Table 2: Path Parameters

Heterogeneity	**# of** `Path(Bandwidth, RTT, PLR)`	
	1	**2**
Low	25Mbps, 10ms, 0	25Mbps, 10ms, 0
Low and Lossy	25Mbps, 10ms, 2%	25Mbps, 10ms, 2%
High	10Mbps, 10ms, 0	40Mbps, 50ms, 0
High and Lossy	10Mbps, 10ms, 2%	40Mbps, 50ms, 2%

5.3 Experimental Results

We validate the effectiveness of PriorityBucket by comparing to existing MPQUIC schedulers. The results are shown in Fig. 3. We can observe that PriorityBucket reduces first rendering time as well as page load time of the web pages in different scenarios, which benefits user experiences. In the high-lossy scenario when loading Bing, the first rendering time and page load time can be reduced up to 34% and 35%. The reduction of page load time mainly attributes to the pre-allocation of paths. This mechanism avoids the streams compete for the fast path in a greedy fashion, causing burst transmissions on the fast path. The first rendering time is decreased due to the prioritized and earlier completed critical streams (HTML, CSS and JavaScript). Comparing the scenarios with and without packet loss, we observe that the scenarios with packet loss experience more fluctuation for repeated experiments. This shows that packet losses will inevitably affect the performance of all the schedulers.

For the two web pages, the time differences between first rendering time and page load time are less evident in loading Bing. This is because the number of page objects that make up YouTube is a lot more than that of Bing, especially the images. When the paths are heterogeneous, our scheduler benefits more significantly. It is because we schedule paths for a stream based on its size. This mechanism reduces stream contention when path heterogeneity exists — we can allocate more portion of a big stream on paths with higher bandwidth, and more portion of a small stream on paths with lower RTT.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed an MPQUIC scheduler aiming at reduce first rendering time in page loading when using HTTP/2. PriorityBucket uses priority bucket in each path to guarantee the prioritization of critical streams. A proof-of-concept implementation and experiments proved the effectiveness of PriorityBucket, showing that it can respectively reduce the first rendering time and page load time up to 34% and 35% in high path heterogeneity when loading Bing. In the future work, we are interested in exploring the adaptability of PriorityBucket when facing varying network conditions. Meanwhile, further evaluations can be made in real world cases.

ACKNOWLEDGEMENTS

This work is partially supported by the National Natural Science Foundation of China (grant numbers 61520106005, 61761136014) and the National Key Research and Development Program of China (grant number 2017YFB1010001). The Corresponding Author is Zhiyong Liu (zyliu@ict.ac.cn)

REFERENCES

[1] Yehia El-khatib, Gareth Tyson, and Michael Welzl. Can SPDY really make the web faster? In *2014 IFIP Networking Conference, Trondheim, Norway, June 2-4, 2014*, pages 1–9, 2014.
[2] 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.
[3] Alexander Rabitsch. Evaluation of packet schedulers for multipath quic. In *Master Thesis, Karlstad University, Karlstad, Sweden, June 19, 2018*, 2018.
[4] Hypertext transfer protocol version 2 (http/2). https://tools.ietf.org/html/rfc7540.
[5] 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.
[6] 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.
[7] Tobias Viernickel, Alexander Frömmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. Multipath QUIC: A deployable multipath transport protocol. In *2018 IEEE International Conference on Communications, ICC 2018, Kansas City, MO, USA, May 20-24, 2018*, pages 1–7, 2018.
[8] Quic: A udp-based multiplexed and secure transport, draft-ietf-quic-transport-13. https://tools.ietf.org/html/draft-ietf-quic-transport-13.
[9] Hang Shi et al. STMS: improving MPTCP throughput under heterogeneous networks. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018.*, pages 719–730, 2018.
[10] Alexander Rabitsch, Per Hurtig, and Anna Brunström. A stream-aware multipath QUIC scheduler for heterogeneous paths. 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.
[11] Rasmus Suhr Mogensen. Reliability enhancement for lte using mpquic in a mixed traffic scenario. In *Technical Report, Aalborg University, Aalborg, Denmark, June*

7, 2018, 2018.

[12] Jing Wang, Yunfeng Gao, and Chenren Xu. A multipath QUIC scheduler for mobile HTTP/2. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking, APNet 2019, Beijing, China, August 17-18, 2019.*, pages 43–49, 2019.

[13] Xiang Shi, Lin Wang, Fa Zhang, and Zhiyong Liu. Fstream: Flexible stream scheduling and prioritizing in multipath-quic. In *25th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2019, Tianjin, China, December 4-6, 2019*, pages 921–924. IEEE, 2019.

[14] 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.

[15] 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.

[16] Neal Cardwell et al. BBR: congestion-based congestion control. *ACM Queue*, 14(5):20–53, 2016.

[17] Nikhil Handigol et al. 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.

[18] Recorded dependency graph and web objects of famous web pages. http://wprof.cs.washington.edu/spdy/tool/.