Multipath QUIC Schedulers under the Microscope

Andreea Lupu, Daniel Petri*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: andreea.lupu@tum.de, petriroc@net.in.tum.de

***Abstract*—Multi-homed devices, such as phones, can use WiFi and cellular networks on the same device. Most of the HTTP/2 traffic comes from mobile devices, through video streaming and web pages. Efficient traffic prioritization is crucial for loading web content. The high latency can negatively impact user experience and the revenue of companies. Improving performance requires reducing the transmission delays across heterogeneous network paths, which are caused by inefficient scheduling decisions and retransmission.**

Multipath QUIC (MPQUIC) is the solution researched in this paper. Depending on its application, MPQUIC is used to either mitigate the inter- and intra-stream Head-of-Line blocking by leveraging different schedulers, path, stream, uplink, and downlink schedulers, or to improve the Quality of Experience in video streaming by modifying the congestion control mechanisms. Our study compares different implementations of MPQUIC in mobile environments, exposing similarities, differences, and their research gaps.

***Index Terms*—MPQUIC, MPTCP, head-of-line blocking, mobile networks**

1. Introduction

Latest developments in mobile networks and web technologies have increased the volume and the complexity of the Internet traffic, especially for video streaming and web browsing. Efficient transmission is critical, as it can damage the revenue and the user satisfaction. Google reports that an additional 500ms at the Page Load Time (PLT) decreases up to 20% of the user engagement, while Bing reports that a similar delay can lead to a 1.2% decrease in their revenue [1]–[3]. The extra delays can arise from the enlarged complexity of the web pages, including the increased number of resources, as well as the size.

Video streaming also suffers from similar challenges. Delays lead to buffering, interruptions, and maybe never seeing a segment again [4]. Although the network technologies are performant for end-to-end devices, it would be advantageous to extend these technologies to a multipath strategy due to the diversity of the network interface cards (NICs) in mobile devices.

The most common protocol currently used on the Internet is HTTP over the Transmission Control Protocol (TCP). Multipath TCP (MPTCP) was developed to enable a single connection to use different paths simultaneously [5]. However, both TCP and HTTP suffer from Head-Of-Line (HOL) blocking; HTTP due to scheduling and TCP due to retransmission, both leading to high latencies in web page completion times [6].

To address the issues with TCP, the QUIC protocol has been developed. It enables multiple byte streams to be sent independently over a single session, preventing congested streams from blocking each other [6]. As with TCP, QUIC should support a diversity of NICs, and therefore, *Multipath QUIC (MPQUIC)* was introduced.

2. Background

This section explains the concepts used in MPQUIC implementations. We start with the problem that MPQUIC tries to mitigate, moving to MPTCP, and lastly explaining the original MPQUIC protocol.

HOL Blocking. The HOL blocking challenge is categorized into two parts: *inter-stream* and *intra-stream* [6]. The inter-stream HOL blocking occurs when processing one resource is delayed by another resource. HTTP/1.1 over TCP allows only one resource to be transmitted at a time, which can cause blocking at the application layer in case of retransmissions or delays. Although HTTP/2 reduces this issue through stream multiplexing, blocking still occurs due to the in-order transmission of TCP. QUIC allows the usage of multiple independent streams over a single session to mitigate the transport-level HOL blocking, but it may suffer from application-layer blocking when the stream prioritization and scheduling are handled poorly. The intra-stream HOL blocking occurs in QUIC when the data within one stream must be delivered in order, and one packet is lost, and subsequent packets are blocked.

MPTCP. Lim et al. [7] highlight that alternating between WiFi and cellular networks makes it difficult for TCP to utilize both interfaces. MPTCP divides a single connection into multiple subflows across all end-to-end interfaces, enabling aggregated bandwidth and allowing WiFi and cellular connections to be combined. The scheduler uses the minimal Round Trip Time (minRTT) estimation, but it lacks the path heterogeneity examination, which is common in mobile networks. MPTCP selects the path with the lowest RTT and an available congestion window, and ensures in-order delivery at the connection level, not only at the subflow level. To cover the packet loss, MPTCP includes a retransmission method that passes lost packets from a slow to a fast path.

ECF. The original MPTCP fails to take advantage of heterogeneous paths with different bandwidths and delays.

 doi: 10.2313/NET-2026-05-1_16

MinRTT transmits data over the lowest RTT path only when its congestion window allows, without accounting for the impact on the flow completion time (FCT). Lim et al. [7] introduce the *Earliest Completion First (ECF)* scheduler for MPTCP to minimize the FCT by selectively scheduling data across available paths based on the estimated completion time. Unlike minRTT, ECF considers both the bandwidth and the number of bytes to be sent at the connection level to determine whether the FCT is reduced or increased if it would send data on a specific subflow.

MPQUIC. To enhance QUIC for multi-homed devices, MPQUIC was developed. De Coninck et al. [8] present two main goals: adapting to connection errors and coordinating resources on different paths. The authors include a Path ID in the header of the packet so that the path can be identified. An additional scope is to help MPQUIC maintain information such as RTT, congestion state, and packets lost in case the address changes over a path. MPQUIC enables different paths to have different packet number spaces so that a receiver can acknowledge packets from different paths. The path manager is used to create and delete paths; a path is created over each interface during the handshake. If one or more paths are active, the sender must select the path to send data on. The packet scheduler does the selection. MPQUIC uses the same scheduler as MPTCP, namely minRTT. The difference is that MPQUIC also decides which type of frame is sent on a particular path. Moreover, MPQUIC enables a flexible retransmission strategy by sending a window update frame on all paths. This way, it avoids retransmission on the same path. MPQUIC allows a fair resource distribution. Implementations may integrate the Opportunistic Linked Increases Algorithm (OLIA) congestion control mechanism [9]. If the subflows are not sharing a bottleneck, all congestion windows are treated separately; otherwise, they are grouped, and the congestion control operates in multipath mode [4]. The use of minRTT as a scheduler created the need to develop different scheduling mechanisms, such as those described in the next section.

3. MPQUIC-Scheduler Design

This section details the MPQUIC schedulers: SA-ECF [2], HBES [10], CAPS [1], MPQUIC-SBD [4]. Its scope is to highlight their different methods.

3.1. SA-ECF

To overcome the asymmetry in bandwidth and delay, Rabitsch et al. [2] propose an MPQUIC algorithm, called *Stream-Aware ECF (SA-ECF)*. This scheduler is based on ECF and ensures a fair allocation of the aggregated bandwidth, utilizing the prioritization provided by HTTP/2 through the dependency tree. The algorithm has two main components: a *stream scheduler* and a *path scheduler*, as highlighted in Figure 1. First, the stream scheduler assigns a fair share of the aggregated bandwidth to active streams based on the received HTTP/2 information about the resources. Next, the path scheduler decides which path each stream should utilize to transmit data. The results show a lower stream completion time (SCT) and well handling of heterogeneous paths [2].

Fair Stream Scheduler. As input, it takes the HTTP/2 dependency tree, where each node is a stream. On its input, it uses the Weighted Round Robin (WRR) algorithm as follows: each parent node shares sending opportunities with its children according to the weights received from the dependency tree. An opportunity is a share of aggregated bandwidth that a stream can send data on. If a node is active, it will take the opportunity to send data. If a node is inactive, meaning it has no data to share, it passes its opportunity further to its children. Each stream receives a concrete number of opportunities. If one stream is unable to send its data, SA-ECF does not give the stream another opportunity to send its share until it is its turn again. The retransmission and control messages have a higher priority and are therefore sent before stream frames, ensuring fairness within each scheduling round [2].

Earliest Completion First. ECF determines which path the data should be sent on [2]. SA-ECF implements a per-stream ECF mechanism aiming to minimize the FCT. The scheduler determines for each stream whether it is better to wait for a faster path that is currently occupied or to send the data through a slower path. The decision depends on the remaining amount of data to be sent and the available bandwidth of each path. The transmission waits for the faster path if and only if the FCT would be lower than the transmission over the slower path. This decision significantly reduces the completion time.

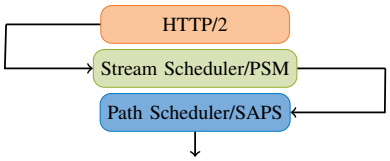

Figure 1: Abstract design of SA-ECF and HBES [10].

3.2. HBES

Xing et al. [10] discuss two challenges of MPQUIC: the transmission of out-of-order (OFO) packets over multiple paths and the prioritization of streams. The authors propose a solution, the *HoL Blocking Eliminating Scheduler (HBES)*, to address inter-stream and to avoid intra-stream HOL blocking. The solution consists of two components: the *Priority-based Stream Manager (PSM)* and the *Stream-aware Arrival-time-based Path Selector (SAPS)*, as displayed in Figure 1. First, the PSM is used to determine the stream that would send data and the number of packets to send. Next, SAPS is responsible for selecting the path that would result in the lowest arrival time. The results show a reduction in the SCT and a good send buffer optimization [10].

Priority-Based Stream Manager. PSM handles the HTTP/2 prioritization mechanism for the streams based on the dependency tree [10]. First, the parent node must complete its transmission before the children can send. The client usually sets priorities for each stream in the

 doi: 10.2313/NET-2026-05-1_16

dependency tree. To schedule the stream, the authors have developed a scattered WRR (SWRR), which limits the number of packets a stream can send in each round, thereby preventing exhaustion of the sending window. Afterwards, PSM determines the number of packets a stream is able to send in a round and sets a token number for each stream accordingly. When the parent node has finished, all the children are visited in the order of establishment, and the token number of packets is sent to each child. PSM also ensures that each stream receives its opportunities, thereby eliminating the inter-stream HOL blocking based on those opportunities.

Stream-Aware Arrival-Time-Based Path Selector. SAPS determines the path for the sent packets [10]. The decision is based on the actual RTT, throughput, and the send buffer capacity. It calculates the total arrival time (queuing time + flight time) for each packet on each path and chooses the shortest one, even if it involves queuing. It constantly observes the RTT and throughput, enabling it to adapt to changes. It implements a send buffer at the path-level to hold the payload scheduled for paths that are occupied, allowing complete packets to be formed at a later time and for the payload to be sent in a concrete order. SAPS will use the path with the lowest RTT until its buffer starts to fill up, then it will continue with the next smallest. In case all congestion windows are complete, SAPS will hold, but only the scheduled packets will be affected.

3.3. Context-aware MPQUIC

Wang et al. [1] address the suboptimal PLT issue that arises using heterogeneous paths in mobile networks. MPTCP sends the acknowledgement (ACK) frames back on the same path on which the data arrived, which can lead to OFO packets. Moreover, the original MPQUIC does not consider the delays and dependencies introduced by inter-object downloading. The solution proposed to combat these challenges is the *Context-Aware MPQUIC Packet Scheduler (CAPS)*. CAPS consists of two schedulers, one for the uplink (client to server) and one for the downlink (server to client), and a mechanism that determines the order of stream transmissions based on priority and size, thereby improving the SCT accordingly. Figure 2 illustrates the context-aware MPQUIC architecture at an abstract level. The results show a decrease of up to 8.5% in the PLT and up to 12.9% in the SCT [1].

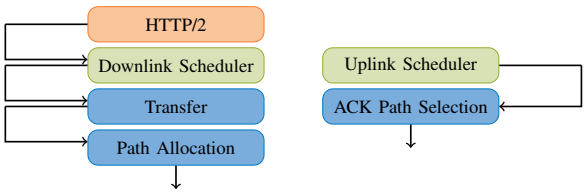

Figure 2: CAPS Architecture. Two separate schedulers, uplink and downlink, improve the PLT.

Stream-Aware Downlink Scheduler. The scheduler takes the path characteristics and stream priority into account [1]. To optimize the delivery of the resources, the scheduler utilizes the dependency tree. There is a parent-child relation between the nodes, where each parent applies a scheduler based on a modified WRR (MWRR) on its children. It calculates the completion time of streams using a fair WRR on the stream length and dependency tree to ensure fairness and avoid starvation. The priority scheme of Firefox was used, which introduces empty, non-resource nodes to gather other nodes and gives them different weights [6]. After the completion time is calculated and the transfer order is determined, it will send the streams exclusively. The following step is to ensure that all paths complete a stream data transfer at once. The SCT is calculated based on the bandwidth, RTT, and queuing time. The paths are taken in RTT order and transmit packets until they are full. Then, it selects streams based on priority with sending limits, and at last, it updates the stream state.

Latency-aware Uplink Scheduler. The context-aware mechanism sends the ACK frames on the path that has the lowest delay, reducing the RTT [1]. To determine the fastest route for ACK, the protocol sends regularly duplicated ACK packets on all uplink paths. The server monitors the identical ACKs, and it maintains the information as valid. Based on this information, it chooses the path for the ACK frame. The authors inserted a new control frame, called CHANGE_RETURN_PATH_FRAME, to elect an ACK route for each path. The ACK's Path ID does not play any role in the congestion window. This way, MPQUIC removes the HOL blocking and ensures retransmission on rapid paths.

3.4. MPQUIC-SBD

In the paper [4], Paiva et al. discuss the problem encountered in video streaming environments. They propose *MPQUIC Shared Bottleneck Detection (MPQUIC-SBD)*, which makes the congestion control adapt dynamically when multiple subflows share a bottleneck. The MPQUIC-SBD scope is to increase the fairness and to balance the bandwidth utilization of video streams. The solution handles the stream multiplexing on different subflows. The results of the research show a 13% increase in the throughput for the non-shared bottlenecks scenarios and 90% accuracy in the SBD [4].

SBD Decision. The SBD protocol is implemented at the receiver side, although both the client and the server must be able to use SBD [4]. The receiver handles the metric relying on the timestamp provided by the sender. SBD considers only those packets from the same connection that have this timestamp to perform the One-Way-Delay (OWD) calculations. Based on the calculated metrics, SBD groups the subflows into sharing or not sharing a bottleneck and makes its decision based on the majority vote of the last 10 observations.

New QUIC Frames and Congestion Control. MPQUIC-SBD extends the QUIC frames by adding the following attributes for stream frames: Path ID, Loss Count, and Timestamp, while it adds the SBD Epoch ID and Group fields in the ACK frames [4]. The new fields of the stream frames are required for computing the OWD and the loss

metrics without creating overhead on the network. The fields are used for congestion control to scale the congestion window of each subflow. The OLIA algorithm is responsible for coupling subflows with a shared bottleneck and decoupling subflows without, enabling the congestion control adaptation.

The Sparsity of Video Transmissions. The video transmission can have a sparse density and a pause period between the fragments, leading to noise introduction [4]. Paiva et al. propose a solution to eliminate the noise issue that uses the timer received from the sender to calculate the time elapsed, the OWD, and the SBD metrics. To further eliminate the noise, they implement a filter to take out the noise caused by sparsity.

4. Comparison

In this section, we will compare the implementations. Table 1 summarizes the main similarities and differences between the four implementations.

TABLE 1: MPQUIC Scheduler Comparison

Features	**SA-ECF** [2]	**HBES** [10]	**CAPS** [1]	**SBD** [4]
Algorithm	WRR	SWRR	MWRR	minRTT
OFO	Medium	Strong	Low	Low
D. Tree	✓	✓	✓	✓
Stream-aware	✓	✓	✓	×
Real-World	×	✓	✓	×

From an architectural design perspective, HBES and SA-ECF have a similar structure. Both implementations first select the stream based on the HTTP/2 priorities, and then the path is decided. SA-ECF decides the path at the stream level, while HBES decides at the packet level [2], [10]. Both implementations also used the HTTP/2 dependency tree mechanism, where the parents are responsible for resource allocation, and the children are allowed to send data only when the parent node is inactive [2], [10]. Moreover, both implementations give a concrete number of opportunities to each stream to send its fair share of data.

A significant difference is the scheduling algorithm that was used. SA-ECF uses the WRR, while HBES uses a scattered WRR, which limits the number of packets each stream can send in each round. Another difference lies in how they handle OFO packets. SA-ECF focuses on avoiding slow paths. HBES instead tries to eliminate OFO packets by estimating the total completion time. As a result, HBES mostly maintains the in-order arrival of the data, leading to better management of the OFO data and eliminating HOL [10]. At the same time, SA-ECF utilizes its buffer to the maximum in path asymmetry scenarios [10].

Despite their differences, Xing et al. [10] show that both implementations have a similar result for FCT when using small streams. HBES performs better than SA-ECF because it is more successful in eliminating the HOL delays. Both protocols aim for a low maximal FCT through the fair share of its stream. In a serial experiment, SA-ECF and HBES outperform other protocols, such as RR or Lowest-RTT-First, by avoiding high-latency paths for small streams. By doing so, the FCT is reduced. In a parallel experiment, HBES successfully reduces the FCT of high-priority streams, but the tradeoff between ordering and throughput gets highlighted because the entire throughput is reduced.

The paper [10] also compares the overall performance, and its results show that both protocols have the best performance, achieving a reduction of 70% in scenarios with a lot of asymmetry, with HBES performing slightly better than SA-ECF. SA-ECF waits for the lowest preferred path, while HBES is able to schedule data for unavailable paths due to the buffer for sending; therefore, HBES is able to achieve faster transmission at the stream level.

Neither SA-ECF nor HBES addresses dynamic switching of scheduling mechanisms to make it more adaptable in the case of different scenarios [2], [10]. SA-ECF focuses on stable networks, while HBES focuses on mobile networks, but the two scenarios are not always exclusive. Plus, neither SA-ECF nor HBES schedulers are testing the interaction with HTTP/3. Also, SA-ECF does not evaluate its implementation in a real-world scenario; it is on an emulated network. HBES is using a small testbed. All these would be good points to research further.

HBES and SA-ECF are not the only implementations making MPQUIC aware of the context. Wang et al. [1] develop CAPS, a solution to make MPQUIC context-aware for the HTTP/2 mobile traffic. The solution uses the HTTP/2 dependency tree as well as the stream priorities. The goal is to eliminate HOL and reduce PLT in mobile networks.

Wang et al. [1] derive a WRR-based algorithm, but they modify it. First, the decision is taken based on the stream priority and length. Afterwards, it sends the stream exclusively, improving the SCT across multiple paths. This is different from HBES and SA-ECF, which focus on fair-based multiplexing scheduling.

Moreover, CAPS enables different schedulers for up- and downlink [1]. The downlink ensures the order and the shares for the data that is sent, while the uplink scheduler ensures that ACK frames take the fastest way to the destination. This is a unique difference from HBES and SA-ECF because it reduces the RTT. HBES and SA-ECF send the ACK on the same path that it came from [2], [10].

Similar to SA-ECF, CAPS is making the scheduling decision per stream. Wang et al. [1] highlight the importance of the real-world scenario of the user in motion. Despite their research, CAPS does not explicitly measure the OFO packets and their improvement for their implementation, as the OFO packets play an important role in HOL improvement.

Paiva et al. [4] take a different approach to handling the multipath improvement for QUIC. Their solution focuses on the coupled congestion control of separate paths. By integrating SBD in MPQUIC, the approach decides whether or not to couple subflows based on the OWD metrics. MPQUIC-SBD does not consider the stream and path scheduling, nor where to send the ACK frames. The scheduler maintains the minRTT path scheduler.

MPQUIC-SBD is testing its implementation in an emulated network environment. Paiva et al. [4] use video streaming to highlight the importance and the results of the created algorithm. Although the results of MPQUIC-SBD are significant, the authors do not consider the HTTP/2

prioritization. The stream selection is not handled at all, while the path selection is not improved. This results in a research gap. Their primary focus is to improve the Quality of Experience and the throughput rather than the multipath diversity. Another research gap is that the MPQUIC-SBD is not evaluated under motion.

5. Conclusion and future work

This paper presents a detailed comparison between four MPQUIC schedulers. All the implementations have proven good results in mitigating the initial problem and are proof that a further improvement of the original MPQUIC is needed.

While some of the implementations focus on making MPQUIC aware of streams and mitigating the HOL problem, others try to improve the congestion control of the subflows. Both of them are required in the mobile networks and must be researched further.

As Google and Bing demonstrated, HOL blocking and additional delays have a huge negative impact on the business and must be improved. Although this paper is tackling the improvement of the traffic in mobile networks, it only discusses HTTP/2. For QUIC, there is a new HTTP version created, HTTP/3, that replaces the dependency tree prioritization of HTTP/2 with the Extensible Prioritization Scheme [11]. Future work should research how the MPQUIC schedulers interact with HTTP/3.

References

[1] J. Wang, Y. Gao, and C. Xu, "A multipath quic scheduler for mobile http/2," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking*, ser. APNet '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 43–49. [Online]. Available: https://doi.org/10.1145/3343180.3343185

[2] A. Rabitsch, P. Hurtig, and A. Brunstrom, "A stream-aware multipath quic scheduler for heterogeneous paths," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 29–35. [Online]. Available: https://doi.org/10.1145/3284850.3284855

[3] E. Schurman and J. Brutlag, "Performance related changes and their user impact," in *velocity web performance and operations conference*, 2009.

[4] T. W. d. P. Paiva, S. Ferlin, A. Brunstrom, O. Alay, and B. Y. L. Kimura, "A first look at adaptive video streaming over multipath quic with shared bottleneck detection," in *Proceedings of the 14th ACM Multimedia Systems Conference*, ser. MMSys '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 161–172. [Online]. Available: https://doi.org/10.1145/3587819.3590982

[5] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8684/

[6] R. Marx, T. De Decker, P. Quax, and W. Lamotte, *Resource Multiplexing and Prioritization in HTTP/2 over TCP Versus HTTP/3 over QUIC*, 11 2020, pp. 96–126.

[7] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Ecf: An mptcp path scheduler to manage heterogeneous paths," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 147–159. [Online]. Available: https://doi.org/10.1145/3143361.3143376

[8] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 160–166. [Online]. Available: https://doi.org/10.1145/3143361.3143370

[9] Y. Liu, Y. Ma, Q. D. Coninck, O. Bonaventure, C. Huitema, and M. Kühlewind, "Managing multiple paths for a QUIC connection," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-multipath-18, Dec. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/18/

[10] Y. Xing, K. Xue, Y. Zhang, J. Han, J. Li, D. S. L. Wei, R. Li, Q. Sun, and J. Lu, "A stream-aware mpquic scheduler for http traffic in mobile networks," *Trans. Wireless. Comm.*, vol. 22, no. 4, p. 2775–2788, Apr. 2023. [Online]. Available: https://doi.org/10.1109/TWC.2022.3213638

[11] K. Oku and L. Pardue, "Extensible Prioritization Scheme for HTTP," RFC 9218, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9218