Multipath QUIC: A Deployable Multipath Transport Protocol

Tobias Viernickel, Alexander Frömmgen, Amr Rizk, Boris Koldehofe, Ralf Steinmetz
KOM — TU Darmstadt
{firstname.lastname}@kom.tu-darmstadt.de

***Abstract*—QUIC is the emerging transport layer protocol, providing encrypted, stream-multiplexed, low-latency data transfer. In this paper, we propose multipath-enabled QUIC (MPQUIC) to leverage multiple network interfaces, such as WiFi and LTE on today's mobile devices. We show how our MPQUIC design conceptually evolves beyond existing multipathing protocols, such as MPTCP, as it provides fine-grained stream-to-path scheduling, reduced head-of-line blocking, and faster subflow establishment.**

We present an userland implementation of MPQUIC that is deployable without operating system changes. Our evaluation results show that MPQUIC increases throughput in comparison to traditional QUIC, TCP and even the currently de facto multipath transport protocol MPTCP. First real world measurements confirm that MPQUIC is deployable in the Internet to reduce download times. Moreover, we show that MPQUIC's conceptual advantages over MPTCP efficiently reduce head-of-line blocking in heterogeneous environments. With multipathing support, QUIC is ready to become the universal stream transport protocol in today's Internet.

I. Introduction

QUIC is an encrypted, multiplexed low-latency transport protocol providing reliable in-order data transfer that is designed to improve web performance for HTTPS [14]. Implemented on top of UDP, QUIC circumvents deployment obstacles as experienced by recent transport protocol innovations. In contrast to transport protocols such as Multipath TCP (MPTCP) [5] and the Stream Control Transmission Protocol (SCTP) [23], QUIC does not require changes to the operating system, making it easily deployable within applications in the userland. Unlike TCP and its evolution MPTCP, QUIC natively multiplexes application streams on a single connection (Figure 1). This supersedes multiplexing HTTP/2 streams on a single TCP stream. First existing Internet-scale QUIC deployments confirm QUIC's design decisions, showing high performance and marginal middlebox interference [14]. With the recently initiated IETF standardization [8], QUIC, which started as a TCP replacement to transport HTTP/2, is becoming a universal transport protocol.

Many of today's communication devices possess multiple network interfaces, e.g., WiFi and LTE. A multipath-enabled QUIC would be able to leverage multiple network interfaces and become *the universal* stream transport protocol. Today's de facto multipathing transport protocol MPTCP already demonstrates the potential of using multiple paths, e.g., for mobile devices [3] and in data centers [18]. However, MPTCP struggles with a number of difficulties due to its primal design based on TCP. In order to remain compatible with TCP and to reduce middlebox interference, MPTCP uses a complex set of TCP options, including a second sequence number space and additional checksums. Furthermore, as extension to TCP, which is typically implemented in the operating system, MPTCP requires operating system support. Finally, out-of-order data arrival that is caused, e.g., by heterogeneous network paths, interferes with the strict in-order delivery of MPTCP, blocking already received packets at the receiver from being processed.

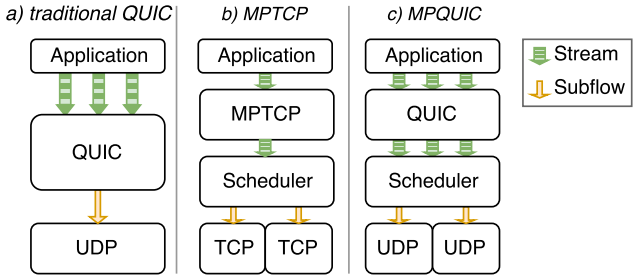


Fig. 1: Network stack comparison of QUIC, MPTCP and MPQUIC. QUIC multiplexes application streams on a single UDP flow, whereas MPTCP splits a single stream on multiple TCP subflows. MPQUIC combines both features by multiplexing application streams on multiple UDP subflows.

In this paper, we *pave the way for Multipath QUIC*. We present the design and implementation of a corresponding QUIC extension, denoted as *MPQUIC* (Figure 1, c). Our evaluations show that MPQUIC increases throughput by utilizing multiple paths. Thus, MPQUIC reduces download times compared to the big competitors QUIC, TCP and MPTCP. We show that the first-class support for multiple streams reduces head-of-line blocking in multipath environments with heterogeneous paths—a well-known issue of MPTCP. Unlike recent transport protocols such as SCTP and MPTCP, our MPQUIC implementation is deployable in the userland without operating system changes. Additionally, MPQUIC is compatible with traditional QUIC, providing the same benefits over TCP such as a faster handshake and full encryption.

The remainder of this paper is structured as follows. Based on a discussion of existing multipath transport approaches (Section II), Section III presents relevant background on QUIC. In Section IV, we present the design and implementation of MPQUIC. Section V presents a detailed evaluation while Section VI concludes the paper.

978-1-5386-3180-5/18/$31.00 ©2018 IEEE

II. Related Work: Multipath Transport Protocols

The related work comprises a number of transport protocols with multipath support [5, 12, 13, 15, 21, 22]. These protocols typically provide mechanisms to establish additional paths and schedule traffic between paths. Many of these protocols, such as MPTCP [5] and SCTP [23], require operating system support. This slows down initial protocol adoption and innovations after deployment, in particular, as mobile devices are known to have long OS update intervals.

Some multipathing protocols, e.g., MMPTCP [13] and MPUDP [15], are designed for specific network environments, such as data centers or virtual private networks, respectively.

Multipath TCP (MPTCP) is the most widely known multipathing transport protocol [5]. Existing MPTCP deployments showed the potential benefits of multipathing [2]. However, the design of MPTCP suffers due to its TCP roots, i.e., the complex TCP headers with an additional sequence number space and checksums. These TCP headers mitigate most but not all middlebox interference, a common problem of recent protocol innovations, such as MPTCP, SCTP and DCCP [11, 20]. In contrast to Multipath TCP, MCTCP [21] adds a shim layer on top of multiple TCP connections and encodes most control information in the payload.

In this paper, we argue that MPQUIC has conceptual design advantages over previously proposed multipath transport protocols. QUIC is easily deployable and enables an integration of multipath control information with minimal middlebox interference. QUIC combines necessary semantics, e.g., of multiple application flows and the corresponding HTTP/2 priorities, to schedule packets in heterogeneous environments.

Alternative MPQUIC Design: De Coninck et al. [4] propose an own multipath extension for QUIC which was developed independently and concurrently with this work. Inspired by Multipath TCP, both designs show comparable concepts, e.g., with regard to the packet number spaces and path management. While De Coninck et al. [4] provide a comprehensive MPQUIC design, we additionally contribute real-world measurements and discuss MPQUIC scheduling opportunities which go beyond MPTCP.

III. QUIC Background

In this section, we briefly introduce today's single paths QUIC (*Quic UDP Internet Connections*), a novel transport protocol originally developed by Google to speed up HTTP-based web traffic. The main objective of QUIC is to provide a deployable, secure and low latency protocol [14]. With the currently ongoing standardization by the IETF, QUIC is becoming a general purpose transport protocol.[1] In the following, we introduce important aspects of QUIC, which are reflected in the general packet format (Figure 2).

UDP-based: QUIC solves deployability issues by running on top of UDP. Since UDP is an established transport protocol, the operation of QUIC is usually not disturbed by middleboxes, as shown with wide-scale deployments in today's Internet [14, 20]. Further, running on top of UDP enables userland implementations of QUIC with unmodified operating systems. Therefore QUIC implementations can be easily upgraded and shipped with applications.

[1]In this paper, we refer to Google's established QUIC version, which slightly diverges from the IETF version [1, 8].

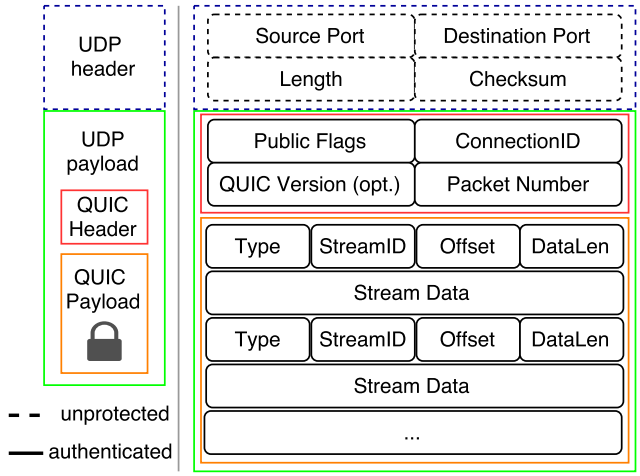


Fig. 2: Example of a QUIC packet [8]: A QUIC packet consists of an authenticated header and encrypted payload frames of (potentially) multiple streams.

Encryption: QUIC always provides encryption and authentication, except for handshake and reset packets. By authenticating the header, QUIC prevents middleboxes from manipulations and limits protocol ossification [14]. The QUIC payload is always encrypted (Figure 2).

To achieve QUIC's low latency objective, cryptographic information are shared together with transport information in one handshake. After a client becomes aware of the initial keys of the server, it is possible to establish new QUIC connections with zero round trip time (0-RTT) data delay.

Streams: QUIC provides stream-multiplexing, that is the ability to handle several request/response pairs concurrently on a single connection by using multiple streams. Each stream is identified by a `StreamID` and possesses an own sequence number space in the `Offset`-header to reduce inter-stream dependencies (Figure 2). Thus, packet loss does not block the entire connection, but only the affected streams which are transmitted in the lost packet. This efficiently avoids head-of–line blocking between streams.

To open a stream, a client or a server sends data, marked with an unused `StreamID`. The appearance of the unused `StreamID` implicitly triggers the creation of a new stream at the receiving side. This reduces the required handshake massages as well as protocol complexity and allows a 0-RTT stream establishment.

QUIC internally multiplexes streams using *frames*. A QUIC packet may contain multiple frames of potentially different streams as well as additional control frames.

Connection Establishment: The left part of Figure 3 depicts the QUIC connection establishment in case client and

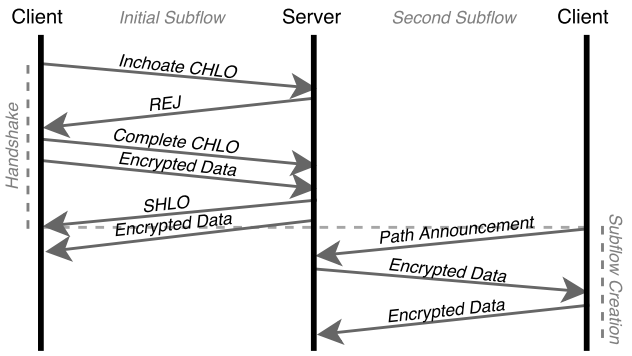


Fig. 3: Timeline of subflow establishment in MPQUIC. The left side depicts the initial QUIC connection establishment. The announcement of any additional subflow, which can be started after receiving or sending the SHLO respectively, is shown on the right side.

server did not communicate so far. The client sends an inchoate hello (CHLO) message to the server. The server answers with a reject (REJ) containing a signed server configuration, a certificate chain and a source address token [14].

Based on the REJ message, the client sends a complete CHLO message, containing its ephemeral cryptographic information. This information is used later on for fast connection establishment with a 0-RTT latency. The server answers with a server hello (SHLO) message containing own ephemeral cryptographic information which completes the handshake. Based on the ephemeral cryptographic information, both sides calculate keys for encrypting subsequent data [14]. In contrast to TCP, the QUIC connection is not identified by the IP-address/port tuple, but uses a `ConnectionID` to enable connection migrations between IP-address/port tuples.

IV. Multipath QUIC Design

The MPQUIC design calls for many decisions from subflow establishment to scheduling. In the following, we present our design which remains compatible with today's QUIC.

A. Subflow Establishment

We introduce *announcement* packets to manage subflow establishment. Therefore, the *announcement* packet, which has the public `MultipathFlag` set, is sent from the IP-address and port (respectively the network interface) to be added (Figure 3). Note that the `MultipathFlag` is already reserved for multipathing in the QUIC draft without further specification [8]. The receiver of an *announcement* packet matches the *announcement* packet to an existing QUIC connection using the `ConnectionID`. Note that packets from an unknown address with a valid `ConnectionID` but unset `MultipathFlag` trigger connection migration.

Figure 4 shows the corresponding subflow states. As the subflow initiator is unaware of the successful subflow establishment until it receives data on the respective subflow, it must

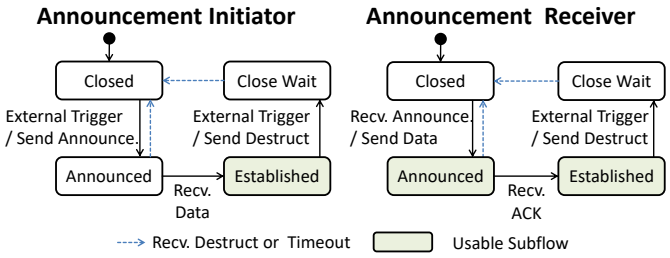


Fig. 4: State machine for establishing additional subflows.

postpone subflow usage till it receives data on the subflow. The announcement receiver, however, can use the announced subflow *immediately* for sending data. This conforms with the behavior of the traditional QUIC connection establishment. Thus, MPQUIC boosts sending data from the server to the client in typical HTTP scenarios with the establishment of an additional subflow after a half RTT.

B. Packet Numbers and Stream Offsets

QUIC relies on packet numbers and stream offsets for reliable data transfer. It avoids ambiguity of fresh and retransmitted packets using unique packet numbering [14]. Accordingly, we propose to use an individual packet number space per subflow. This is particularly important to distinguish and detect packet loss in heterogeneous multipath environments with substantially different RTTs. We found QUIC's existing stream management, i.e., the separation of packet numbers and stream data offsets, sufficient for MPQUIC, as it enables flexible packet scheduling and avoids dependencies at the receiver which might cause head-of-line blocking.

C. Controlling Packet Injection

The injection of packets on subflows depends on four components, *i)* the flow control, *ii)* the congestion control, *iii)* the loss detection, and *iv)* the packet scheduler.

Flow Control: To ensure that the receiver buffer does not overflow, QUIC uses two flow control mechanisms, *i)* a stream- and *ii)* a connection-level flow control. We adopted this concept for MPQUIC with adapted RTT-statistics. Based on the RTT-statistics the flow controller determines the offset of the window updates (the amount of data which can be sent) and its sending interval. Here, the QUIC flow controller makes use of the minimum smoothed RTT of the individual paths resulting in smaller offsets and shorter intervals.

Congestion Control: We propose a per subflow congestion control, which adjusts the amount of data that can be transmitted on this subflow per RTT. The congestion control of the subflows might be coupled to achieve fairness on shared bottlenecks with disjoint paths [19].

Loss Detection and Retransmission: MPQUIC has to detect packet loss and trigger according retransmissions. Therefore, each subflow implements its own loss detection and retransmission schema. The retransmission timeouts are based on the subflows round-trip time estimates. After loss detection, the packet scheduler is in charge of the retransmission.

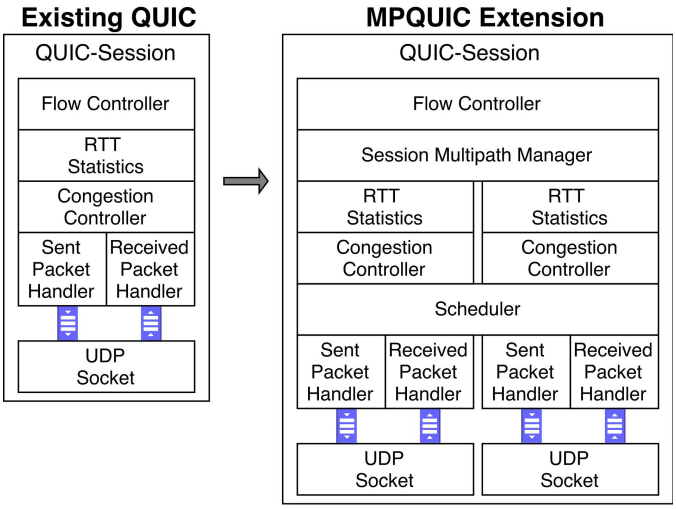


Fig. 5: The design of an existing QUIC implementation and its MPQUIC extension.The `Session Multipath Manager` keeps subflow specific components. The `Scheduler` decides on which subflow to send data.

Packet Scheduling: Our MPQUIC design provides a single scheduling instance which *schedules packets from streams on subflows*. Thus, the scheduler composes packets out of the queued data streams and assigns these packets on subflows. This enables a fine-grained scheduling, considering both subflow properties and stream requirements. The scheduler, for example, has to ensure that a subflow has not exhausted its congestion and flow control window and the stream has not saturated its flow control window. The fine-grained scheduling further allows retransmissions on arbitrary subflows, i.e., stream data of lost packets might be combined to new packets together with fresh stream data.

We propose a Lowest-RTT-Subflow-First scheduler for MPQUIC, which is also the default scheduler of the Linux Kernel MPTCP implementation [16]. Based on previous experiences with MPTCP scheduling [6, 7, 17], we further envision a rich variety of scheduling flavors to consider stream and subflow preferences, e.g., to optimize retransmission handling in accordance with these preferences.

D. Encryption

MPQUIC uses the same security mechanisms as traditional QUIC. Data of all subflows is encrypted using the cryptographic information obtained during the connection setup.

E. Implementation

We implemented MPQUIC based on the existing open-source *quic-go*[2] implementation. Figure 5 illustrates the major changes required for MPQUIC. In quic-go, the core packet processing component is the `QUIC-Session` with a unique `ConnectionID`. Packets with the same `ConnectionID` pass the same handler instances and controllers.

[2]https://github.com/lucas-clemente/quic-go

Our previously introduced MPQUIC design has a per subflow packet number space and RTT-statistics, as well as a decoupled congestion control. Hence, MPQUIC requires certain components on a per subflow basis, i.e., the `Sent` and `Received Packet Handler`, `Congestion Controller`, `RTT-Statistics` and a `UDP-Socket`. The `Session Multipath Manager` handles subflow creation and keeps track of subflow specific components. Whenever a packet is processed by the `QUIC-Session`, the `Session Multipath Manager` provides the subflow based components. Accordingly, in our MPQUIC implementation the subflow of a packet determines the instance of handlers and controllers which process a packet in addition to the `ConnectionID`. Finally, we implemented a scheduler which decides on which subflow to send upcoming data. The scheduler composes packets based on the available data per stream and chooses the according subflow for this packet.

F. Discussion

The prevalent design of QUIC with its given header flags, i.e., the `ConnectionID`, enables a natural extension. We found MPQUIC to be easy to integrate in established QUIC implementations, as it essentially adds subflow management and packet scheduling components.

MPTCP Comparison: In comparison to MPTCP, our design of MPQUIC provides a few unique selling propositions. The MPQUIC subflow establishment, for example, requires only a half RTT, whereas MPTCP requires a full TCP handshake. The awareness of different streams enables a more fine-grained scheduling. In comparison, as Multipath TCP only provides a single stream, upper layer protocols such as HTTP/2 have to multiplex different application-specific streams on a single MPTCP stream. This implies that MPTCP packets are treated in a FIFO order throughout the transport protocol stack. The MPTCP scheduler only *sees* a single byte stream which makes it oblivious with respect to application stream priorities. As a consequence, MPTCP packets of different application streams have implicit dependencies, e.g., a high priority HTTP response for JavaScript may be blocked (due to head-of-line blocking) by a previously scheduled large image transfer. Figure 6 shows an illustration of MPTCP's FIFO treatment compared to MPQUIC's stream awareness.

V. Evaluation

In this section, we provide a detailed experimental evaluation based on our MPQUIC implementation. We show the deployablility of MPQUIC in the wild and compare the performance of MPQUIC with traditional QUIC, traditional TCP, and the established MPTCP for different traffic patterns and network characteristics using Mininet [10]. If not stated otherwise, all subflows have a bottleneck capacity of 10 Mbps and 44ms round-trip time (RTT) in the Mininet emulations.

A. Real World Deployability

To prove the deployablility and increased throughput in real-world environments, we run our MPQUIC implementation on

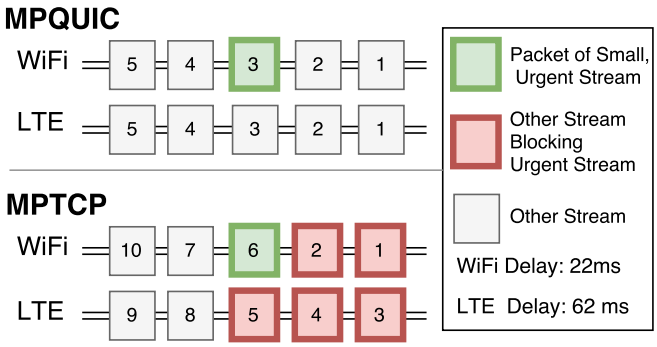


Fig. 6: Illustration of packets of different streams on subflows. In MPQUIC, previous packets containing other streams do not block the urgent stream. MPTCP's fully ordered delivery causes packets of other subflows to block the urgent stream.

a notebook with WiFi and LTE interfaces, while the latter runs via USB tethering over a Nexus 5 device. The server runs on an Amazon AWS virtual machine. We found that MPQUIC is able to leverage multiple paths in real world environments, and thereby significantly reduces download times (Figure 7).

For future work, we envision a massive real world evaluation, which also implements a proper solution for using network interfaces concurrently. With respect to the scope of this paper, we use Mininet in the following to concentrate on a systematic evaluation in reproducible environments.

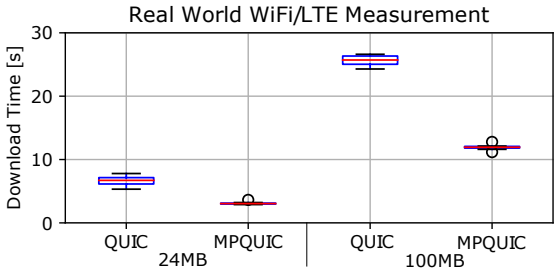


Fig. 7: Comparison of QUIC (over WiFi) and MPQUIC in a real world setup. MPQUIC reduces the download time by using an additional LTE path.

B. MPQUIC Speedup with Homogeneous Paths

For a first proof of concept, we compare the performance of traditional QUIC with MPQUIC in a setup with two homogeneous paths. As QUIC originates in HTTP/2, we use HTTP/2 file downloads of varying sizes for our evaluation.

The evaluation with two subflows (Figure 8) shows that small downloads do not benefit from multipathing. These small downloads are not throughput limited, but suffer from the overhead of the initial QUIC handshake and the HTTP/2 request. For files of 100kB, however, MPQUIC already provides a significant speedup. For larger files, the speedup of the download time nearly reaches the theoretical maximum of a factor of 2. With an increasing number of available paths

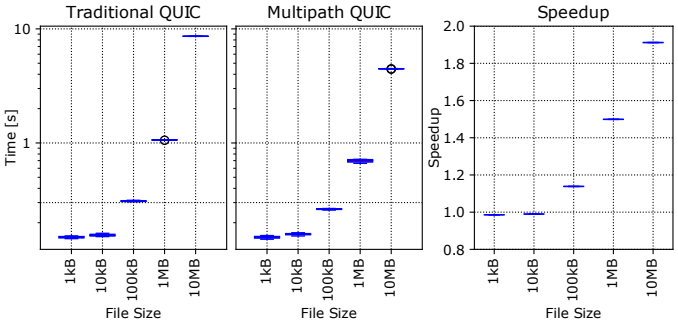


Fig. 8: Comparison of QUIC and MPQUIC in a homogeneous environment. The speedup of MPQUIC increases with the file size while for small files the handshake latency is prevalent.

(and corresponding subflows), MPQUIC further decreases the required download time for large files as shown in Figure 9a.

C. Impact of Path Heterogeneity

Based on the experiments with homogeneous paths, we now analyse the impact of path heterogeneity in terms of bandwidth and RTTs. Here, we consider a scenario with 2 paths.

Bandwidth Heterogeneity: Figure 9b shows the performance of MPQUIC under heterogeneous path capacities, i.e., we set the bandwidth of the first path to $B_1 = 10$ Mbps and vary the bandwidth ratio of the first and the second path B_2/B_1 for a 1MB file and homogeneous round-trip times. The measurements show that MPQUIC benefits from additional paths even if these paths only offer a low bandwidth. In this experiment, adding a low bandwidth subflow with half the capacity of the first subflow reduces the required download time from 1.06s given traditional QUIC (obtained from Figure 8) to 0.86s using MPQUIC.

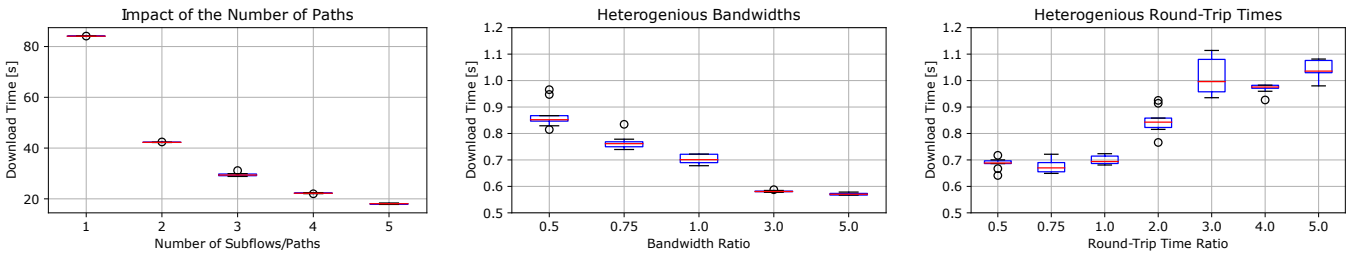
Delay Heterogeneity: Figure 9c shows the performance of MPQUIC given heterogeneous round-trip times of two different paths. Here we fix the RTT of the first path $R_1 = 44$ ms and vary the bottleneck delay D_{2b} of the second path. The resulting RTT ratio is $(2 \times D_{2b})/(R_1 - 4ms)$. The additional 4ms result from delays at links apart from the bottleneck.

The measurements show that MPQUIC slightly benefits from an additional low round-trip time subflow. In this two-path example we note that with an increasing RTT ratio MPQUIC behaves more like a traditional single path QUIC.

D. Comparison of MPQUIC, TCP and MPTCP

As we envision QUIC to become a universal transport protocol, we compare MPQUIC with the established state-of-the-art protocols TCP and MPTCP. For the MPTCP evaluations, we relied on the publicly available MPTCP Linux Kernel implementation [16].

Single File Download: A comparison of the download times for the introduced network setup with two subflows reveals that MPQUIC outperforms all considered competitors (Figure 10) in environments with homogeneous paths. For small files (i.e., 10kB and 100kB) the speedup of MPQUIC compared to MPTCP is higher than for larger files (i.e., 1MB).



(a) Impact of an increasing number of subflows respectively paths.

(b) Impact of the path capacity heterogeneity (bandwidth ratio).

(c) Impact of the path latency heterogeneity (RTT ratio).

Fig. 9: The impact of the numbers of paths and path heterogeneity on the download time of a 1MB file in MPQUIC

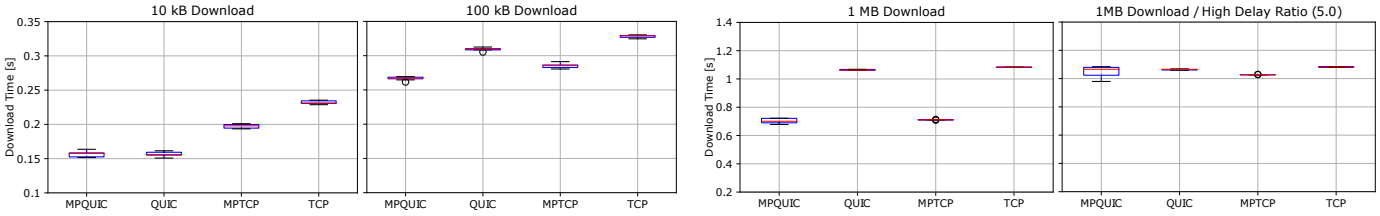


Fig. 10: Comparison of MPQUIC, QUIC, MPTCP and TCP for HTTP/2 file downloads.

We notice that even traditional QUIC is faster than MPTCP when loading a 10kB file. Furthermore, we note that MPTCP suffers from high round-trip time ratios in the same way as MPQUIC (Figure 10, right).

Webpage Download: Next, we compare the page load time with HTTP/2 in combination with QUIC, MPQUIC, TCP and MPTCP. We measured the overall page load time consisting of the handshake, the HTTP/2 request of the initial HTML file, and the HTTP/2 requests for all dependencies, including all inherent files (i.e., JavaScript, CSS and images). Figure 11 confirms our previous measurements, showing that MPQUIC outperforms QUIC, TCP and MPTCP.

As Langley et al. [14] showed performance benefits for QUIC in comparison to TCP, we conclude that the design of QUIC enables its multipath extension to outperform MPTCP, the de facto multipath protocol.

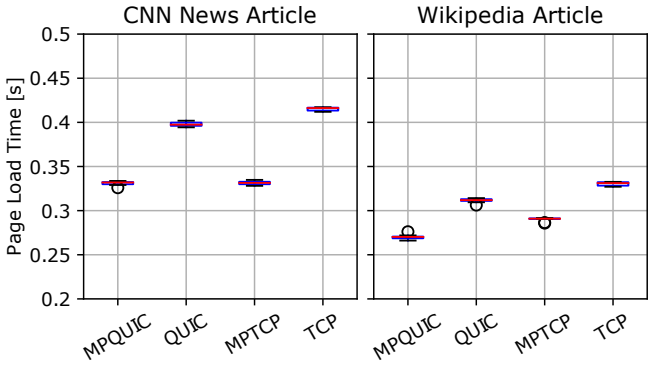


Fig. 11: HTTP/2 page load times with two subflows.

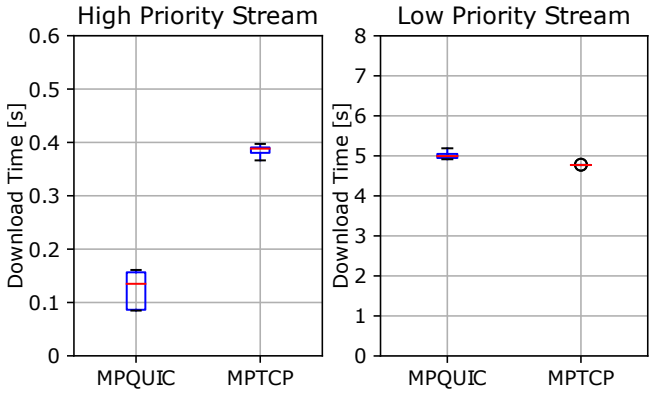


Fig. 12: In contrast to MPTCP, MPQUIC provides the ability to distinguish application specific stream priorities (here in the context of web page JavaScript objects vs. images).

E. Overcoming Stream-Subflow Scheduling Limitations

According to our previous discussion in Section IV-F, we evaluate the design advantages of considering multiple streams of MPQUIC in comparison with MPTCP. Here we consider a request for a small (1KB) high priority file while loading a large (10MB) low priority file. This illustrates a typical web page retrieval scenario, where a small JavaScript file is requested *while* images are being retrieved. Figure 12 shows the download time for the high priority file in a scenario with two heterogeneous paths, i.e., a low delay (22ms) WiFi and a high delay (62ms) cellular path. It is evident that MPQUIC is able to prioritize the sought stream accordingly. Note that in case of MPQUIC the high priority stream is not blocked by the low priority stream at the *receiver* side (Figure 6).

VI. Conclusion

This paper presented MPQUIC, a multipath extension for the emerging QUIC transport protocol. We showed that MPQUIC naturally combines the advantages of QUIC and Multipath TCP. Our measurements showed that MPQUIC increases throughput up to the theoretical maximum speedup. Confirmed by real world measurements, MPQUIC is deployable and reduces download times over the Internet.

In contrast to MPTCP, MPQUIC runs in the userland on top of UDP, making it easily deployable without operating system changes. Additionally, our design of MPQUIC extends the 0-RTT connection establishment known from QUIC to the multipath case providing a performance speedup on subflow establishment in comparison to MPTCP. Moreover, we showed that stream multiplexing in MPQUIC is superior to the established TCP stream abstraction for MPTCP as it reduces head-of-line blocking between application streams by leveraging the different available paths. Most notably, MPQUIC proved that it transfers the advantages of QUIC into the world of multipathing by outperforming MPTCP, and thus taking up a leading position of today's general purpose protocols.

This paper presents the first design for multipath enabled QUIC. We envision emerging research on optimized MPQUIC scheduling, detailed real world measurement studies, as well as, further application scenarios such as video streaming, comparable to [3, 7, 9, 17] for Multipath TCP.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of the projects C2 and B4 in the Collaborative Research Center (SFB) 1053 MAKI.

References

[1] The Chromium Projects: QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic.

[2] O. Bonaventure and S. Seo. Multipath TCP Deployments. *IETF Journal*, 12(2):24–27, 2016.

[3] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. A measurement-based study of Multipath TCP performance over wireless networks. In *IMC*, pages 455–468, 2013.

[4] Q. De Coninck and O. Bonaventure. Multipath QUIC: Design and Evaluation. In *CoNEXT*, 2017.

[5] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, 2013.

[6] A. Frömmgen, T. Erbshäußer, T. Zimmermann, K. Wehle, and A. Buchmann. ReMP TCP: Low Latency Multipath TCP. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1–7, 2016.

[7] A. Frömmgen, A. Rizk, T. Erbshäußer, M. Weller, B. Koldehofe, A. Buchmann, and R. Steinmetz. A Programming Model for Application-defined Multipath TCP Scheduling. In *ACM/IFIP/USNIX Middleware*, 2017.

[8] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. *IETF, draft-hamilton-early-deployment-quic-00*, 2017.

[9] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath. In *CoNEXT*, 2016.

[10] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible Network Experiments using Container-based Emulation. In *CoNEXT*, pages 253–264. ACM, 2012.

[11] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure. Are TCP extensions middlebox-proof? In *Hot Middleboxes*, pages 37–42. ACM, 2013.

[12] J. Heuschkel, A. Frömmgen, J. Crowcroft, and M. Mühlhäuser. VirtualStack: Adaptive Multipath Support through Protocol Stack Virtualization. In *International Network Conference (INC)*. Lulu.com, 2016.

[13] M. Kheirkhah, I. Wakeman, and G. Parisis. MMPTCP: A multipath transport protocol for data centers. In *INFOCOM*. IEEE, 2016.

[14] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *SIGCOMM*, pages 183–196. ACM, 2017.

[15] D. Lukaszewski and G. Xie. Multipath transport for virtual private networks. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*, 2017.

[16] C. Paasch and S. Barre. Multipath TCP in the Linux Kernel. available from http://www.multipath-tcp.org.

[17] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental Evaluation of Multipath TCP Schedulers. In *SIGCOMM Workshop on Capacity Sharing*, pages 27–32. ACM, 2014.

[18] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with Multipath TCP. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 266–277, 2011.

[19] C. Raiciu, M. Handly, and D. Wischik. Coupled congestion control for multipath transport protocols. RFC 6356, 2011.

[20] J. Rosenberg. UDP and TCP as the New Waist of the Internet Hourglass. *Work in Progress*, 2008.

[21] M. Scharf and T.-R. Banniza. Mctcp: A multipath transport shim layer. In *GLOBECOM*. IEEE, 2011.

[22] R. Steward, P. Amer, and J. Iyengar. Concurrent Multipath Transfer Using SCTP Multihoming Over Independant End-to-End Paths. In *Transactions on Networking*. IEEE/ACM.

[23] R. Stewart. Stream Control Transmission Protocol. RFC 4960, 2007.