Multipath QUIC: Design and Evaluation

Quentin De Coninck[*]
UCLouvain, Belgium
quentin.deconinck@uclouvain.be

Olivier Bonaventure
UCLouvain, Belgium
olivier.bonaventure@uclouvain.be

ABSTRACT

Quick UDP Internet Connection (QUIC) is a recent protocol initiated by Google that combines the functions of HTTP/2, TLS, and TCP directly over UDP, with the goal to reduce the latency of client-server communication. It can replace the traditional HTTP/TLS/TCP stack and the IETF has chartered a working group to standardize it. QUIC encrypts all data and most protocol headers to prevent interferences from middleboxes.

Motivated by the success of Multipath TCP (MPTCP), we design Multipath QUIC (MPQUIC), a QUIC extension that enables a QUIC connection to use different paths such as WiFi and LTE on smartphones, or IPv4 and IPv6 on dual-stack hosts. We implement MPQUIC as an extension of the `quic-go` implementation. We evaluate the benefits of QUIC and MPQUIC by comparing them with TCP and MPTCP in a variety of settings. MPQUIC maintains MPTCP's benefits (aggregation benefit, network handover). Without packet losses, while performance of single-path TCP and single-path QUIC are similar, MPQUIC can outperform MPTCP. In lossy scenarios, (MP)QUIC is more suited than (MP)TCP.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Transport protocols**; *Application layer protocols*; Cross-layer protocols;

ACM Reference format:
Quentin De Coninck and Olivier Bonaventure. 2017. Multipath QUIC: Design and Evaluation. In *Proceedings of CoNEXT '17: The 13th International Conference on emerging Networking EXperiments and Technologies, Incheon, Republic of Korea, December 12–15, 2017 (CoNEXT '17),* 7 pages.
https://doi.org/10.1145/3143361.3143370

1 INTRODUCTION

Innovation in the network and transport layers of the TCP/IP protocol suite has been rather slow during the past decades. It took roughly twenty years to design, implement and deploy IPv6 [13]. Despite the proliferation of middleboxes [22], TCP continues to be incrementally improved [14] with recent extensions including Multipath TCP [17] or TCPCrypt [3]. New transport protocols such as SCTP [46] generated interest [5], but are still not widely deployed.

Given the prevalence of middleboxes on the Internet, innovation in the transport layer must take a different approach. QUIC (Quick UDP Internet Connection) [10, 44] is an interesting solution to this problem. It was initially proposed by Google to speed up page downloads. QUIC includes from day one the cryptographic mechanisms that are required to authenticate the server and negotiate encryption keys, so QUIC encrypts both the data and almost all the header fields. This prevents middleboxes from interfering with QUIC and ensures that the protocol will not be ossified by deployed middleboxes. QUIC uses a flexible packet format and leverages the reliability and congestion control mechanisms of modern TCP stacks. Finally, QUIC supports different streams that prevent head-of-line blocking when downloading different objects from a single server.

One missing feature of QUIC is the ability to exploit the different paths that exist between a client and a server. Today's mobile devices such as smartphones have several wireless interfaces and users expect to be able to combine them easily. Furthermore, a growing fraction of hosts are dual-stack and the IPv4 and IPv6 paths between them often differ and have different performance [2, 29, 30]. Multipath TCP (MPTCP) [17] is a recent TCP extension that addresses this problem since it enables a TCP connection to send data over different paths. MPTCP can aggregate the bandwidth of the different paths that it uses [41]. This is beneficial on smartphones that need to combine WiFi and cellular [7]. On smartphones, MPTCP enables faster handovers and improves user experience [35]. Apple has recently announced [8] that all iOS11 applications will be able to use MPTCP. QUIC cannot currently support those use cases because it uses a single UDP flow between the client and the server. QUIC connection migration allows moving a flow from one address to another. This is a form of hard handover. Experience with MPTCP on smartphones [4, 11] shows that multipath provides seamless handovers.

In this paper, we first describe the basic principles of QUIC in Section 2. We then build upon the lessons learned with MPTCP [41] and propose in Section 3 extensions to QUIC to enable it to simultaneously operate over multiple paths. Our design remains clean and simple thanks to the flexibility of the QUIC protocol. We implement Multipath QUIC inside the QUIC open-source implementation written in go [9] and compare in Section 4 its performance with MPTCP in a wide range of scenarios using Mininet [21]. Section 5 summarizes our work and presents future work.

2 QUIC BACKGROUND

Large web companies and Content Distribution Networks make huge efforts to improve the performance and the security of web protocols. QUIC [20, 23, 24] is a recent proposal initiated by Google and embraced by many others that collapses the functions of the classical HTTP/2, TLS and TCP protocols into a single application

[*] FNRS Research Fellow

CoNEXT '17, December 12–15, 2017, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5422-6/17/12…$15.00
https://doi.org/10.1145/3143361.3143370

layer protocol that runs over UDP [10]. QUIC is already widely deployed, Google recently estimated that QUIC represents 7% of the total Internet traffic and the IETF is actively working on standardizing a version of QUIC [23]. The design of QUIC has been inspired by earlier work in the transport layer [18, 46]. An important design choice of QUIC is that QUIC messages are almost entirely encrypted to prevent the ossification that middleboxes cause on protocols like TCP [22].

QUIC is a connection-oriented protocol. Each *QUIC connection* starts with a secure handshake [1] and is identified by a *Connection ID* (CID). During the secure handshake, hosts negotiate the version of QUIC that will be used. The combination of version negotiation and encryption allows QUIC to easily evolve regardless of middleboxes. Each QUIC packet contains a small unencrypted *public header* containing a few flags, the CID, the Packet Number (PN) and an encrypted payload. Each host maintains separate PNs for sending and receiving data. PNs are monotonically increasing on a connection. When data is retransmitted, it is placed in a packet with a higher PN than the original one. This simplifies several transport functions by removing the ambiguity of multiple retransmitted packets with the same PN that affects round-trip-time estimation and loss recovery in TCP. The encrypted payload of a QUIC packet contains one or more *frames*. *Frames* carry data and control information while QUIC packets act as their "containers". They are detailed in [23]. The main ones are briefly summarized below.

QUIC supports several streams like SCTP [46] as required for HTTP/2. The STREAM frame contains a Stream ID, a byte offset and the payload associated to that stream. The ACK frame is a kind of generalization of TCP and SCTP's SACK blocks. ACK frames contain ACK blocks that provide detailed information on the received frames. They also include an ACK delay field enabling the peer to accurately estimate round-trip-time, even if ACK frames are delayed. The WINDOW_UPDATE frame[2] is used to advertise the receive window of the peer. It only appears in some packets, unlike TCP that places a window field in all packets.

Most of the discussions on QUIC occur currently within the IETF and on the related mailing lists. As of October 2017, few scientific articles have analyzed QUIC or its performance. Megyesi et al. compare QUIC, SPDY and HTTP [32]. They analyze page load times and report that each protocol has some advantages over the others in some environments. Carlucci et al. performed a similar study with HTTP/1.1 and an earlier version of QUIC [6]. Langley et al. reported QUIC represents more than 30% of the egress traffic at Google, a large fraction being induced by mobile devices [28]. Kakhki et al. [25] identified some performance issues with the Chromium implementation of Google QUIC.

3 MULTIPATH QUIC

There are two main motivations for adding multipath capabilities to a protocol like QUIC. The first one is to pool resources of different paths to transport the data for a single connection [47]. This is important for large transfers on multihomed devices such as smartphones, but can also allow dual-stack hosts to automatically select the best path when the quality of the IPv4 and IPv6 paths differ.

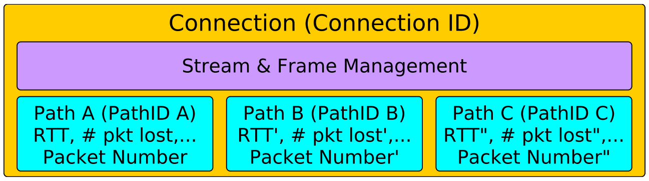

Figure 1: High-level architecture of Multipath QUIC.

Another motivation is the resilience to connectivity failures. On dual-homed hosts with wireless interfaces, such as smartphones, one wireless network can fail at any time and users expect that their applications will immediately switch to the other one without any visible impact [8, 35]. These use cases are well covered by Multipath TCP [4, 8]. The remainder of this section describes the design of Multipath QUIC (MPQUIC) whose high-level architecture is depicted in Fig. 1.

Path Identification. Hosts need to agree on a way to identify the different paths used. A first solution would be to make those paths implicit by sending ranges of packet numbers over a particular path. However, paths can exhibit heterogeneous characteristics with very different delays. Because the packet number in the public header is not encrypted, middleboxes in the network can see it and might decide to drop packets with lower packet number than the highest seen on a connection. Such a device placed in front of a sever using multiple paths might break the slowest path. MPQUIC takes the explicit approach by including in the public header of each packet the Path ID on which it was sent. The presence of the Path ID also allows MPQUIC to use multiple flows when a remote address changes over a particular path, e.g., due to NAT rebinding. In such cases, path information (such as round-trip-time or packets lost) and congestion state can be kept unchanged.

Reliable Data Transmission. QUIC uses encrypted STREAM frames to send data. These frames contain a stream identifier and a byte offset. This information is sufficient to enable a receiver to reorder the STREAM frames that it receives over different paths. However, the acknowledgement is per-packed based and reordering could affect packets sent on different paths due to network heterogeneity. With a single packet number space, this can lead to huge ACK frames containing many ACK blocks. To cope with this, each path has its own packet number space, as shown on Fig. 1. By combining the Path ID and the packet number in the public header, MPQUIC exposes the paths to middleboxes. MPQUIC also adds a Path ID field to the ACK frame. This enables a receiver to acknowledge QUIC packets that have been received on different paths. Our implementation returns the ACK frame for a given path on the path where the data was received, but since it contains the Path ID, it is possible to send ACK frames over different paths. Notice that reusing the same sequence number over different paths might have a detrimental impact on security, as the cryptographic nonce will be reused. To mitigate this, a first solution is to restrict that a sequence number could be used only once over all paths. Another possibility is to involve the Path ID in the nonce computation, such that it is not possible to get the same nonce across different paths.

[1] The currently deployed handshake [31] will be replaced by TLS 1.3 [42].
[2] It will be replaced by MAX_DATA and MAX_DATA_STREAM frames [23].

Path Management. A QUIC connection starts with a secure handshake. Like QUIC, MPQUIC performs the cryptographic handshake over the first path. There are probably opportunities to leverage the availability of several paths for the secure handshake, but we leave this work to future. MPQUIC uses a *path manager* that controls the creation and deletion of paths. Upon handshake completion, it opens one path over each interface on the client host. An important difference compared to MPTCP is that MPQUIC can directly use a new path by placing data in the first packet. MPTCP requires a three-way handshake before being able to use any path [17]. Since MPQUIC allows both hosts to create paths, paths created by the client (resp. the server) have an odd (resp. even) `Path ID` to avoid `Path ID` clashes. However, our implementation does not currently use server-initiated paths because clients are often behind NATs or firewalls that would block them. To enable, e.g., a dual-stack stack server to advertise its IPv6 address to the client over an IPv4-initiated connection, MPQUIC includes an `ADD_ADDRESS` frame that advertises all the addresses owned by a host. This frame can be reliably exchanged at the beginning of a connection or when addresses change. Given that this frame is encrypted, it does not suffer from the security concerns of the `ADD_ADDR` option in MPTCP [17]. MPQUIC also enables hosts to get a global view about the active paths' performances over a connection through the `PATHS` frame. It contains `Path IDs` of the paths considered as active by the sending host and statistics such as the estimated path round-trip-time. It can be used to detect underperforming or broken paths and can thus speed up the handover process in mobility scenarios.

Packet Scheduling. As soon as several paths are active, a MPQUIC sender needs to select over which path each packet will be transmitted. This selection is performed by the *packet scheduler*. Several packet schedulers have been proposed and analyzed for Multipath TCP [16, 33, 36]. For MPQUIC, our starting point is the default scheduler used by the MPTCP implementation in the Linux kernel [34]. It relies on the smoothed measured round-trip-time (RTT) and prefers the path with the lowest RTT provided that its congestion window is not already full. MPQUIC uses the same heuristic, but with two differences. First, while MPTCP has to decide which data (either new or reinjected) will be sent on which path, the MPQUIC scheduler also determines which control frame (`ACK`, `WINDOW_UPDATE`, `PATHS`,...) will be sent on a particular path. Since frames are independent of the packets containing them, they are not constrained to a particular path. When a packet is marked as lost, *its frames are not necessarily retransmitted over the same path*, while MPTCP is forced to (re)transmit data in sequence over each path to cope with middleboxes. Retransmission strategy is thus more flexible in MPQUIC than in MPTCP. To prevent receive buffer limitations, the scheduler ensures proper delivery of the `WINDOW_UPDATE` frames by sending them on all paths when they are needed. Second, when a new path starts in MPQUIC, the scheduler does not have an estimation of the path's RTT yet. A first solution would be to ping the new path and wait 1 RTT to obtain this measurement, but MPQUIC would then lose its ability to directly send data on new paths. Another approach would be to use round-robin at the start of the connection and automatically switch to lowest RTT path once RTT estimations are available. However, this approach is fragile when paths exhibit very different delays and hosts could then possibly face head-of-line blocking. Our scheduler duplicates the traffic over another path when the path's characteristics are still unknown. While this induces some overhead, it enables faster usage of additional paths without facing head-of-line issues.

Congestion Control. To achieve a fair distribution of network resources, transport protocols rely on congestion control algorithms. Both single-path TCP (in the Linux kernel) and QUIC (in the `quic-go` and the Chromium[3] implementations) use CUBIC [19]. Using CUBIC in a multipath protocol would cause unfairness [48]. Several multipath congestion control schemes have been proposed [27, 38, 48]. In our MPQUICimplementation, we integrate the OLIA congestion control scheme [27], which provides good performance with MPTCP. The adaptation and the comparison of other multipath congestion control schemes to MPQUIC is left for further study.

Overall, our Multipath extensions to QUIC are simpler and cleaner than the Multipath extensions to TCP [17], while keeping them as deployable as possible. Thanks to the clean support for multiple streams in `STREAM` frames, MPQUIC does not need to specify a new type of sequence number in contrast to MPTCP's DSN. MPQUIC does not need to specify mechanisms to detect or react to middlebox interference given that all frames are encrypted and authenticated. This also reduces the possibility of attacking a QUIC connection in contrast to MPTCP whose security relies on keys exchanged in clear during the initial handshake [17]. Furthermore, thanks to the independence between packets and frames, MPQUIC can spread multiple data streams over multiple paths by design and the packet scheduling is potentially more powerful than MPTCP's one. Finally, the flexibility of QUIC allows us to easily define new types of frames to enhance the protocol.

4 PERFORMANCE EVALUATION

There are several approaches to evaluate the performance of a transport protocol. Many TCP extensions have been evaluated by simulations before being deployed [15, 40, 41]. The QUIC designers deploy improvements on the Google servers and use the collected statistics to tune the protocol [28]. In this paper, we rely on measurements on the Mininet emulation platform [21] with complete (MP)QUIC and (MP)TCP implementations and use (MP)TCP as the baseline. To provide a fair assessment of the compared implementations, we use an experimental design approach similar to the one used for MPTCP [37] and cover a wide range of parameters. This allows a fairer comparison of the multipath benefits than only considering a few well-chosen cases. Real-world evaluation is part of our future work.

4.1 Large File Downloads

Our first scenario is the download of a large file over a single stream. Here, a multihomed host wants to minimize the download time and thus maximize the aggregation of the bandwidth of the available paths.

We consider a multipath network with two multihomed hosts over disjoint paths with different characteristics as shown in Fig. 2. In this case, the performance of multipath protocols is a function of the links' bandwidth, the round-trip-time, the presence

[3] Chromium recently started to use BBR as its default congestion control.

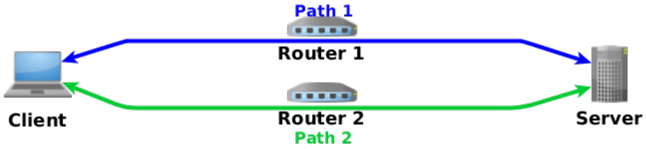

Figure 2: Simple network with two hosts and disjoint paths.

	Low-BDP		High-BDP	
Factor	Min.	Max.	Min.	Max.
Capacity [Mbps]	0.1	100	0.1	100
Round-Trip-Time [ms]	0	50	0	400
Queuing Delay [ms]	0	100	0	2000
Random Loss [%]	0	2.5	0	2.5

Table 1: Experimental design parameters [37].

of bufferbloat (i.e., the queueing delay), and the random packet losses. Our experimental design [37] selects the values of these parameters using the WSP algorithm [45] over the ranges listed on Tab. 1. We group the simulations into four classes: *(low-BDP-no-loss)* environments with a low bandwidth-delay product and no random losses, *(low-BDP-losses)* environments with a low bandwidth-delay product and random losses, *(high-BDP-no-loss)* environments with a high bandwidth-delay product but no random losses and *(high-BDP-losses)* environments with a high bandwidth-delay product and random losses. For each class, we consider 253 scenarios and vary the path used to start the connection, leading to 506 simulations. Each simulation is repeated 3 times for each protocol (TCP, MPTCP, QUIC, MPQUIC) and we analyze the median run. We use the Linux kernel version 4.1.39 patched with MPTCP v0.91 as our baseline. Experiments run on a Intel Xeon E3-1245 V2 @ 3.40 GHz. The VM has two dedicated cores and 2 GB of RAM.

To ensure a fair comparison given that QUIC uses encryption which consumes CPU on our emulation platform, our measurements use `https` over (MP)TCP (TLS 1.2 [43]) or (MP)QUIC (QUIC crypto [31]). Each measurement downloads a 20 MB file in a single stream and the client measures the delay between the transmission of the first connection packet and the reception of the last byte of the file. To ensure a fair comparison, we use CUBIC congestion control with the two single path protocols. Since there is no multipath variant of CUBIC, we use the OLIA congestion control scheme with Multipath TCP and Multipath QUIC. The maximal receive window values are set to 16 MB for both TCP and QUIC.

Low-BDP-no-loss: MPQUIC outperforms MPTCP. Our first metric is the ratio between the delay to receive this file over TCP divided by the time required with QUIC. If the ratio is equal to 1, both protocols are equivalent. If the ratio is larger than 1, TCP is slower than QUIC. Figure 3 provides the CDF of this ratio over all considered scenarios in environments with a low bandwidth-delay product and no random losses. Notice that packet losses due to router buffer overflow can still occur in these environments. When a single path is used, we do not observe a difference between TCP and QUIC. This is expected since in this scenario, the congestion control is the main influencing factor, and both use CUBIC. With multipath, QUIC outperforms TCP in 89% of the considered scenarios. Indeed, MPTCP faces head-of-line blocking more often than MPQUIC. With heterogeneous paths, MPTCP tends to send large bursts of packets on the slow path. This might be related to the ambiguities linked to the estimation of the round-trip-time in the Linux kernel that makes the scheduler prefer the slow path after the fast path is under load, possibly leading to head-of-line blocking. Under those conditions, MPTCP uses the Opportunistic Retransmission and Penalisation (ORP) [41] mechanism that also leads to retransmissions of packets from the slow path over the fast path, limiting overall goodput. Under the same conditions, MPQUIC is less aggressive towards the slow path thanks to its precise path latency estimation and avoids receive buffer limitations thanks to the transmission of the `WINDOW_UPDATE` frames on all the available paths. Without random losses, retransmissions are rare.

Experimental Aggregation Benefit. To better assess the benefits of multipath protocols over single-path ones for specific scenarios, we rely on a modification of the aggregation benefit [26, 37]. Instead of comparing the measured goodput with the sum of the link bandwidths, the *experimental aggregation benefit* compares the sum of the goodputs achieved by single path protocols over the two links with the goodput of the multipath variant. Let C be a multipath aggregation simulation with n paths. G_s^i is the mean goodput achieved by a single-path connection on path i. G_s^{max} is the maximum single-path mean goodput measured over the n paths. The experimental aggregation benefit $EBen(C)$ is given by

$$EBen(C) = \begin{cases} \frac{G_m - G_s^{max}}{(\sum_{i=1}^{n} G_s^i) - G_s^{max}} & \text{if } G_m \geq G_s^{max}, \\ \frac{G_m - G_s^{max}}{G_s^{max}} & \text{otherwise.} \end{cases}$$

An experimental aggregation benefit of 0 indicates that a multipath protocol achieves the same performance as the single path variant over the best path. If multipath achieves a mean goodput equal to the sum of mean goodputs over all paths, then experimental aggregation benefit equals 1. A value of -1 for the experimental aggregation benefit indicates that the multipath protocol failed to transfer data. Because we rely on experimental values, the experimental aggregation benefit can be greater than 1 when the performance of multipath protocols are better than the sum of the performances of the single path variant over each path. In this paper, we compare Multipath QUIC with single-path QUIC and Multipath TCP with single-path TCP.

Low-BDP-no-loss: Multipath is beneficial to QUIC. Figure 4 shows the experimental aggregation benefit over the 253 low-BDP scenarios without random losses. Research with MPTCP [1] has shown that its performance was impacted by the characteristics of the initial path. We thus split the results into two categories depending on whether the connection is created on the best or the worst performing path. Our measurements indicate that MPQUIC seems less affected by the characteristics of the initial path than MPTCP. This is probably related to the duplicate phase of the scheduler and the absence of handshake latency when creating new paths, the only difference between both cases being the QUIC connection establishment latency over the initial path (1 RTT). MPQUIC can reach an experimental aggregation benefit close to 1, even when the connection starts on the less performing path. Independently

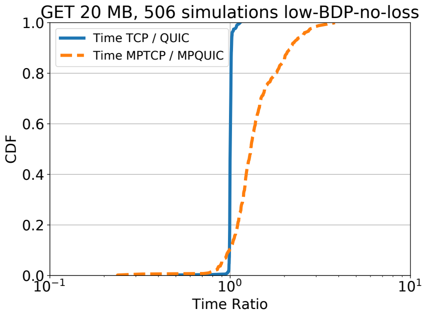

Figure 3: QUIC and TCP exhibit similar performances. On average, Multipath QUIC tends to be faster than Multipath TCP.

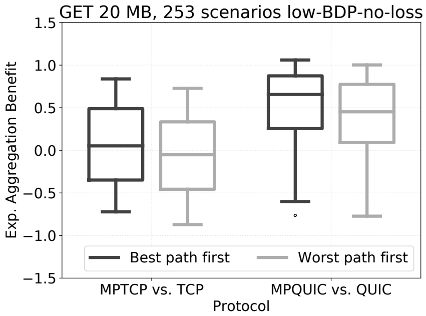

Figure 4: In low-BDP scenarios without random losses using several paths is more beneficial to QUIC than to TCP.

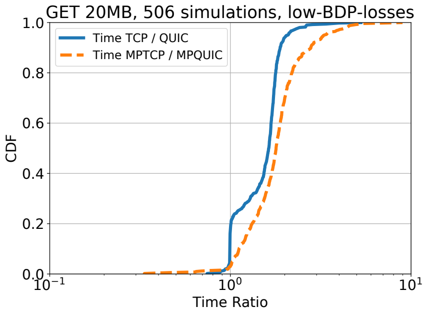

Figure 5: In low-BDP scenarios (Multipath) QUIC reacts faster than (Multipath) TCP to random losses.

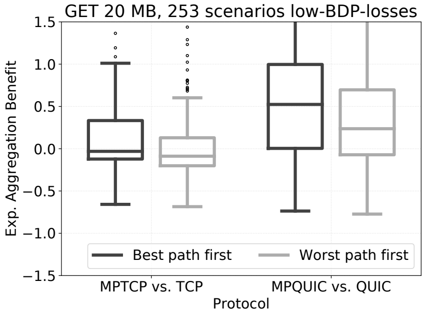

Figure 6: Multipath can still be advantageous for QUIC in lossy environments, though the measured goodput varies more.

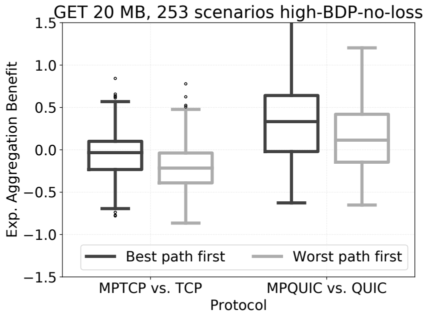

Figure 7: Multipath QUIC remains advantageous for long transfers in high-BDP environments.

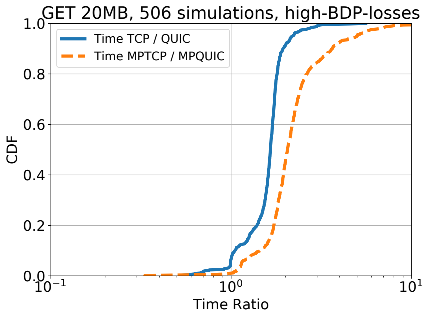

Figure 8: QUIC performs better than TCP in high-BDP environments when there are random losses.

of the initial path, MPQUIC reaches a higher experimental bandwidth aggregation in 77% of our scenarios. For MPTCP, this number drops to 45%. Getting precise estimations of the path latencies helps MPQUIC to balance traffic while avoiding head-of-line blocking.

Low-BDP-losses: (MP)QUIC outperforms (MP)TCP. Random losses, such as those that occur on wireless links, affect the performance of reliable transport protocols. We now analyze the simulations in low-BDP environments with such losses. As shown in Tab. 1, we consider random losses up to 2.5%, which fits in the retransmission rates experienced by the Google's deployment [28]. Figure 5 shows that in those scenarios (MP)QUIC nearly always performs better than (MP)TCP. This difference is mainly due to the ACK frame that can acknowledge up to 256 packet number ranges. This is much larger than the 2-3 blocks than can be acknowledged with the SACK TCP option depending on the space consumed by the other TCP options. Therefore, early retransmits are more effective in QUIC and it suffers less from head-of-line blocking.

Low-BDP-losses: Multipath is still beneficial to QUIC. Figure 6 shows that using multiple paths reduces download time compared to single-path over the best path. Note that the packet scheduling in MPQUIC is not affected by random packet losses given QUIC's round-trip-time estimation. Furthermore, a packet marked as lost is not automatically retransmitted on the same path, unlike MPTCP. When a packet is lost, the OLIA congestion control scheme reduces the congestion window over the affected path. The loss detection mechanisms can enable MPQUIC to take advantage of multiple paths, even if they are lossy.

High-BDP-no-losses: MPQUIC performs well. When considering high-BDP environments, the experimental aggregation benefit of MPTCP decreases, as shown in Fig. 7. In such networks, connections mainly suffer from bufferbloat and head-of-line blocking due to receive window limitations. Using multiple paths with different characteristics does not prevent those issues, and can possibly worsen them if those paths exhibit very different latencies. Due to the additional path establishment delay, MPTCP favors the initial path since it has more time to grow its congestion window and is thus more fragile to bufferbloat. Because all paths can start sending data at connection establishment, congestion windows evolve more fairly and those issues are less visible with MPQUIC. Indeed, regardless of the initial path, multipath is beneficial in 58% of the scenarios with QUIC, while this percentage with TCP drops to 20%.

High-BDP-losses: (MP)QUIC better copes with loss. When adding random loss in high-BDP networks, (MP)QUIC still outperforms (MP)TCP as shown on Fig. 8. The better loss signaling, more precise latency estimation and the fairness in the evolution of the congestion window of paths with (MP)QUIC explain those results.

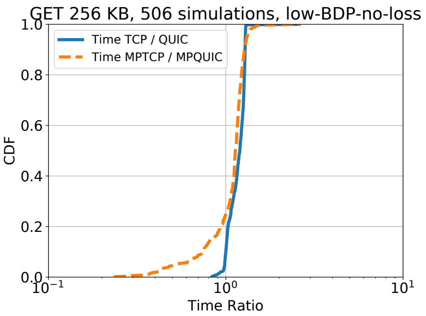

Figure 9: For small transfers, QUIC is faster thanks to its shorter handshake.

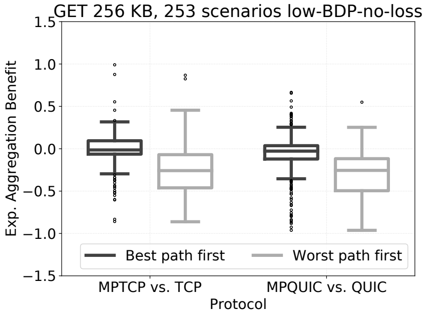

Figure 10: For small transfers, QUIC should remain single-path with heterogeneous paths.

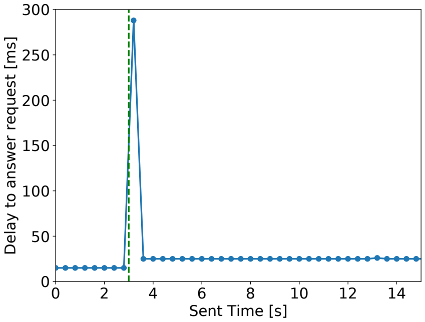

Figure 11: Multipath QUIC is able to perform network handover by design.

4.2 Short File Downloads

Our next scenario considers the download of a 256 KB file on a single stream. Due to space limitation, we only discuss the low-BDP scenarios without random losses.

(MP)QUIC outperforms (MP)TCP. Figure 9 shows that single-path QUIC outperforms HTTPS over TCP. For short transfers, the connection establishment is a non-negligible fraction of the total transfer time. With QUIC, the secure handshake consumes a single round-trip-time. With TLS/TCP, the TCP 3-way handshake and the TLS 1.2 handshake consume together 3 round-trip-times. This delay could be reduced by using the emerging TLS 1.3 [42] and TCP Fast Open option [40].

Multipath is not useful for short transfers. MPQUIC outperforms MPTCP in most situations. However, using a multipath protocol is not really desirable for short transfers, as shown in Fig. 10. When the connection is initiated on the best path, multipath can bring some benefits, as suggested by previous results [12]. However, performance mainly depends on the connection handshake latency and the data transfer might be over before taking full advantage of the multipath opportunities.

4.3 Network Handover

The last studied situation involves request/response traffic over a single connection with random packet loss on the initial path. This corresponds to a smartphone connection to a bad WiFi network and a good cellular network. This is one of the main motivations for adding MPTCP on smartphones [8]. We simulate this scenario with an initial path with a lower latency (15 ms RTT) than the other one (25 ms RTT), and where a client sends 750 bytes long requests every 400 ms. The server immediately replies to each request with a 750 bytes response. Initially, paths are loss-free, and after 3 seconds, the initial path becomes completely lossy.

Figure 11 shows the delay viewed by the MPQUIC client from when the request was triggered until the response is received. Initially, both hosts use the initial path as it exhibits a lower latency. Since the loss event occurs after an answered request, the client is the first host to notice path failure. After facing a RTO, its scheduler will retransmit the request over the slow but functional path. This is because our MPQUIC implementation, like MPTCP in Linux [39], considers a path as *potentially failed* when it experiences a RTO without observing any network activity since last packet transmission. The affected path remains in this state until data is acknowledged on this path. The packet scheduler temporarily ignores *potentially failed* paths. This enables it to quickly decide to use another path when severe losses affect one path. However, when the remote host receives the data that has been retransmitted over the second path, it does not know that a previous copy was sent unsuccessfully over the first path and could decide to respond over this one. This would likely lead to a loss followed by retransmissions and the first path would be marked as *potentially failed* on the remote host. This recovery time could be long for interactive applications. To avoid this additional RTO, the client also adds a PATHS frame in the retransmitted packet indicating that the initial path failed. When the server receives it, it can then directly send the response back to the client without experiencing RTO. The connection can then continue on the functional path.

5 DISCUSSION

Multipath capabilities are important for smartphones and dual-stack hosts. In this short paper, we propose extensions to QUIC that enable this new protocol to use several paths simultaneously. Our extensions remain simple thanks to QUIC's flexible design. We implemented Multipath QUIC in the open-source QUIC implementation in go. Our performance evaluation, over more than a thousand Mininet scenarios that cover a wide range of parameters, shows that Multipath QUIC can provide improved benefits to QUIC than Multipath TCP to regular TCP. They also indicate that (MP)QUIC copes better with packet losses than (MP)TCP.

This work is a first step to the inclusion of native multipath capabilities inside QUIC. Our further work will be to port it to smartphones and dual-stack hosts, and explore its performance through large-scale Internet measurements.

Software artefacts. To ensure the repeatability of our results and allow other researchers to improve Multipath QUIC, we release our modifications to the QUIC implementation written in go, the Mininet images used to perform the simulations and all the benchmark applications that we used. These are available at `https://multipath-quic.org`.

We thank our anonymous reviewers and our shepherd, Adrian Perrig, whose constructive comments have helped us to improve the paper.

REFERENCES

[1] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo. 2014. Deconstructing mptcp performance. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE, 269–274.

[2] R. Beverly, W. Brinkmeyer, M. Luckie, and J. P. Rohrer. 2013. IPv6 alias resolution via induced fragmentation. In *PAM'13*. Springer, 155–165.

[3] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. 2010. The Case for Ubiquitous Transport-Level Encryption.. In *USENIX Security Symposium*. 403–418.

[4] O. Bonaventure and S. Seo. 2016. Multipath TCP Deployments. *IETF Journal* (November 2016).

[5] Ł. Budzisz, J. Garcia, A. Brunstrom, and R. Ferrús. 2012. A taxonomy and survey of SCTP research. *ACM Computing Surveys (CSUR)* 44, 4 (2012), 18.

[6] G. Carlucci, L. De Cicco, and S. Mascolo. 2015. HTTP over UDP: an Experimental Investigation of QUIC. In *SAC'15*. ACM, 609–614.

[7] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. 2013. A measurement-based study of multipath tcp performance over wireless networks. In *IMC'13*. ACM, 455–468.

[8] S. Cheshire, D. Schinazi, and C. Paasch. 2017. Advances in Networking, Part 1. (June 2017). `https://developer.apple.com/videos/play/wwdc2017/707//`.

[9] L. Clemente et al. 2017. A QUIC implementation in pure go. (2017). `https://github.com/lucas-clemente/quic-go`.

[10] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind. 2017. Innovating Transport with QUIC: Design Approaches and Research Challenges. *IEEE Internet Computing* 21, 2 (2017), 72–76.

[11] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure. 2016. A first analysis of multipath tcp on smartphones. In *International Conference on Passive and Active Network Measurement*. Springer, 57–69.

[12] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. 2014. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In *IMC'14*. ACM, 181–194.

[13] A. Dhamdhere, M. Luckie, B. Huffaker, A. Elmokashfi, E. Aben, et al. 2012. Measuring the deployment of IPv6: topology, routing and performance. In *IMC'12*. ACM, 537–550.

[14] M. Duke et al. 2015. A Roadmap for Transmission Control Protocol (TCP) Specification Documents. RFC7414. (February 2015).

[15] K. Fall and S. Floyd. 1996. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *ACM SIGCOMM Computer Communication Review* 26, 3 (1996), 5–21.

[16] S. Ferlin, O. Alay, O. Mehani, and R. Boreli. 2016. BLEST: Blocking Estimation-based MPTCP Scheduler for Heterogeneous Networks. In *IFIP Networking 2016*. IEEE Computer Society, IEEE, Los Alamitos, CA, USA.

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

[18] B. Ford and J. R. Iyengar. 2008. Breaking Up the Transport Logjam.. In *HotNets*. 85–90.

[19] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 64–74.

[20] R. Hamilton et al. 2016. QUIC: A UDP-Based Multiplexed and Secure Transport. (July 2016). Internet draft, draft-hamilton-quic-transport-protocol-00.

[21] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. 2012. Reproducible network experiments using container-based emulation. In *CONEXT'12*. ACM, 253–264.

[22] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. 2011. Is it still possible to extend TCP?. In *IMC'11*. ACM, 181–194.

[23] J. Iyengar and T. M. 2017. QUIC: A UDP-Based Multiplexed and Secure Transport. (June 2017). Internet draft, draft-ietf-quic-transport-04.

[24] J. Iyengar and I. Swett. 2015. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. (July 2015). Internet draft, draft-tsvwg-quic-protocol-00.

[25] A. M. Kakhki, S. Jero, D. Choffnes, A. Mislove, and C. Nita-Rotaru. 2017. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *IMC'17*. ACM.

[26] D. Kaspar. 2012. Multipath aggregation of heterogeneous access networks. *ACM SIGMultimedia Records* 4, 1 (2012), 27–28.

[27] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec. 2012. MPTCP is not pareto-optimal: performance issues and a possible solution. In *CONEXT'12*. ACM, 1–12.

[28] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

[29] I. Livadariu, A. Elmokashfi, and A. Dhamdhere. 2016. Characterizing IPv6 control and data plane stability. In *IEEE INFOCOM 2016*. IEEE, 1–9.

[30] I. Livadariu, S. Ferlin, Ö. Alay, T. Dreibholz, A. Dhamdhere, and A. Elmokashfi. 2015. Leveraging the IPv4/IPv6 identity duality by using multi-path transport. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*. IEEE, 312–317.

[31] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. 2015. How secure and quick is QUIC? Provable security and performance analyses. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 214–231.

[32] P. Megyesi, Z. Krämer, and S. Molnár. 2016. How quick is QUIC?. In *ICC'16*. IEEE, 1–6.

[33] B.-H. Oh and J. Lee. 2015. Constraint-based proactive scheduling for MPTCP in wireless networks. *Computer Networks* 91 (2015), 548–563.

[34] C. Paasch, S. Barre, and et al. 2009-2017. Multipath TCP in the Linux Kernel. (2009-2017). http://www.multipath-tcp.org.

[35] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure. 2012. Exploring mobile/WiFi handover with multipath TCP. In *CellNet'12*. ACM, 31–36.

[36] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. 2014. Experimental evaluation of multipath TCP schedulers. In *SIGCOMM workshop on Capacity sharing*. ACM, 27–32.

[37] C. Paasch, R. Khalili, and O. Bonaventure. 2013. On the benefits of applying experimental design to improve multipath TCP. In *CONEXT'13*. ACM, 393–398.

[38] Q. Peng, A. Walid, J. Hwang, and S. H. Low. 2016. Multipath TCP: Analysis, design, and implementation. *IEEE/ACM Transactions on Networking* 24, 1 (2016), 596–609.

[39] C. Pinedo. 2015. Improve active/backup subflow selection. (January 2015). https://github.com/multipath-tcp/mptcp/pull/70.

[40] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. 2011. TCP Fast Open. In *CONEXT'11*. ACM, 21.

[41] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. 2012. How hard can it be? designing and implementing a deployable multipath TCP. In *NSDI'12*. USENIX Association, 29–29.

[42] E. Rescorla. 2017. The Transport Layer Security (TLS) Protocol Version 1.3. (April 2017). Internet draft, draft-ietf-tls-tls13-20.

[43] E. Rescorla and T. Dierks. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. (Aug. 2008). `https://doi.org/10.17487/RFC5246`

[44] J. Roskind. 2013. QUIC(Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP. *Technical report, Google* (2013).

[45] J. Santiago, M. Claeys-Bruno, and M. Sergent. 2012. Construction of space-filling designs using WSP algorithm for high dimensional spaces. *Chemometrics and Intelligent Laboratory Systems* 113 (2012), 26–31.

[46] Stewart, R. (Ed.). 2007. Stream Control Transmission Protocol. (Sept. 2007). RFC4960.

[47] D. Wischik, M. Handley, and M. B. Braun. 2008. The resource pooling principle. *ACM SIGCOMM Computer Communication Review* 38, 5 (2008), 47–52.

[48] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP.. In *NSDI'11*.