Multiflow QUIC: A Generic Multipath Transport Protocol

Quentin De Coninck and Olivier Bonaventure

***Abstract*—Transport protocols and their multipath variants used to assume that network paths are symmetric and bidirectional. Actually, it is frequent to observe network asymmetries (ADSL, satellite,...) and some network paths can flow packets in only one direction (e.g., due to a firewall,...). This paper proposes to consider the more generic notion of unidirectional flows. Based on them, we design and implement Multiflow QUIC, a variant of QUIC that is aware of network asymmetries to spread data over multiple network paths. Our evaluation shows that this more generic approach is beneficial in asymmetric cases while being equivalent to multipath approaches in symmetric ones.**

I. Introduction

THE Transmission Control Protocol (TCP) is the dominant Internet transport protocol. It provides a reliable data stream between two hosts across the network. A TCP connection uses a fixed given network path denoted by a 4-tuple (IP_{src}, $port_{src}$, IP_{dst}, $port_{dst}$). Thanks to its acknowledgments, TCP includes flow control, congestion control and reliable delivery. All these mechanisms assume that packets can flow in both directions between the communicating hosts.

Nowadays, there is an increasing fraction of heavily connected, mobile devices having two or more network interfaces. When, e.g., a smartphone loses its Wi-Fi connectivity, all the established TCP connections using it are broken, even if the cellular network is available. Therefore, the multipath ability is key for transport protocols to cope with current [1] and future use cases. The first widely deployed multipath transport protocol is Multipath TCP [2]. It enables hosts to combine several TCP subflows for a single data stream. For instance, a smartphone can couple the Wi-Fi network path with the cellular one to either aggregate their bandwidth or to be resilient to network handovers.

Meanwhile, the QUIC protocol [3] emerged as an alternative to the HTTP2/TLS/TCP stack. Previous work [4] showed that QUIC already represented more than 30% of the egress Google traffic. One of its key features is that a QUIC connection is no more bound to a given 4-tuple. This *connection migration* ability was primarily designed to cope with network events such as NAT rebindings, but QUIC can also handle network path changes. Yet, QUIC does not support the simultaneous usage of multiple network paths. We previously proposed Multipath QUIC [5] to address this gap. Its design was influenced by the Multipath TCP one, e.g., it assumes bidirectional network paths and sends acknowledgements back over the same path.

This assumption of symmetrical, bidirectional network paths can be limiting, or even sub-optimal, in some networks. There are network links that provide different bandwidths in upload and download directions. Asymmetric Digital Subscriber Line (ADSL) and satellite Internet providers are typical examples. A recent survey confirms that US network providers often propose asymmetric broadband services [6]. Previous studies [7] showed that the upload and the download flows of a single connection can actually use separate network paths, resulting in different one-way delays. Even if the flow sender $\rightarrow$ receiver uses a high-bandwidth low-delay network path, a connection can be impacted by a receiver $\rightarrow$ sender flow using a low-bandwidth high-delay network path. Indeed, the slow transmission of acknowledgments impacts the throughput of TCP connections [8]. Similarly, keeping acknowledgments on the same subconnections limits the performance of Multipath TCP in asymmetrical networks [9].

Transport protocols should not limit themselves to bidirectional, assumed symmetrical, network paths. To make them aware of link asymmetries, we propose to consider *unidirectional flows*. Such approach is more generic, as a bidirectional path is formed by a couple of unidirectional flows in opposite directions. With this in mind, we design and implement *Multiflow QUIC*, a multipath transport protocol that is aware of unidirectional flows. Compared to Multipath QUIC, Multiflow QUIC gets rid of the bidirectional constraint to fully take advantage of asymmetric, or even unidirectional, network paths. This paper is organized as follows. We

All authors were with the UCLouvain, Belgium. Quentin De Coninck is a F.R.S.-FNRS Postdoctoral researcher. This work was partially supported by the Wallinnov MQUIC project.

Manuscript received October 2, 2020, revised December 10, 2020, accepted January 20, 2021.

first introduce the QUIC protocol and describe its connection migration mechanism (Sec. II). We then explain the design of Multiflow QUIC (Sec. III) and compare its performance with Multipath QUIC in asymmetric network cases (Sec. IV). Finally, we conclude with the key lessons learned with Multiflow QUIC (Sec. V).

II. QUIC

QUIC is a transport protocol which provides the services of the HTTP/TLS/TCP stack atop UDP. It provides built-in secured exchanges with both authenticated and encrypted header and payload. A QUIC connection starts with a TLS 1.3 handshake that generates encryption keys. This built-in encryption enables QUIC to address both privacy and ossification issues that middleboxes cause on TCP [10].

From the network viewpoint, a QUIC packet is split into two parts. The first part is a small cleartext header containing a few flags and the Destination Connection ID, mapping the packet to a given QUIC connection. This identifier makes the QUIC connection independent of the packet's 4-tuple. The second part is encrypted and contains several elements. The first element is a monotonically increasing packet sequence number. Each host maintains distinct sequence number spaces for sending and receiving packets. Within one unidirectional flow, *all* packets carry different packets numbers. In case a packet is lost, its content can be retransmitted in another packet with a higher packet number than the previous one. This simplifies several transport functions, such as latency estimation, by removing the ambiguity of having multiple retransmitted packets with the same packet number.

The remaining of the encrypted payload consists in a sequence of *frames*. These are the core QUIC messages carrying data and control information. QUIC packets act as their containers. In particular, when a packet is lost, frames are the messages that are retransmitted, if needed. Frames use a simple Type-Value encoding making it easy to define new ones. The base QUIC version [3] defines about twenty different frames. The STREAM frame carries application data. It contains an ID (identifying the data stream), a byte offset and the associated payload. The ACK frame acknowledges received packets. Compared to TCP, it can easily indicate that specific blocks of packets were received and also includes information for precise latency estimation. Hosts perform flow control by using MAX DATA and MAX STREAM DATA frames. These are the equivalent of the TCP receive window as they advertise the maximum data offset that can be sent over the connection and the stream, respectively. Notice that all the QUIC frames are idempotent, i.e., receiving them more than once does not introduce any ambiguity at the receiver's side.

As previously described, QUIC uses Connection IDs to identify packets belonging to a given connection. Each unidirectional flow has its own set of Connection IDs, i.e., the packets received by a host use a different Connection ID than the ones it sends. The receiver selects the Connection ID that the sender must use to reach it. In other words, the client (resp. the server) sends packets to its peer by using the Connection ID chosen by the server (resp. the client). The Connection IDs used over a given QUIC connection might change over time. Hosts determine the initial Connection IDs during the handshake. Then, each host can provide alternative Connection IDs to its peer by sending NEW CONNECTION ID frames. This enables hosts to mitigate possible traffic correlation by external network observers.

Because it runs above UDP, packets from a QUIC connection may use different 4-tuples during its lifetime. Since QUIC relies on Connection IDs to identify connections, it handles such events by design. However, this connection migration feature raises the concern of the identity of the new remote IP address and port. A malicious host might forge a QUIC packet by spoofing the source IP address of a victim, hoping that the server will flood it. To tackle this issue, QUIC includes a *path validation* process. The host starts sending a PATH CHALLENGE frame containing some random data to the remote address and port to validate. The process succeeds if the host then receives a PATH RESPONSE frame echoing the random data contained in the PATH CHALLENGE. Since frames are encrypted, this process ensures that the host still communicates with the same peer, even if it observes a different 4-tuple.

III. Designing Multiflow QUIC

Multipath TCP [2] and Multipath QUIC [5] consider bidirectional paths. Multiflow QUIC (MFQUIC) takes a more generic approach by handling unidirectional flows (or *uniflows*). This enables MFQUIC to handle network asymmetries by design and opens new use cases for specific unidirectional network paths. We first describe how MFQUIC hosts identify uniflows. We then explain how the Multiflow extension can be negotiated and how it enables hosts to control the number of uniflows. Finally, we describe the mechanisms used by MFQUIC to ensure a reliable data delivery. More details can be found in its IETF draft [11].

A. Identifying Uniflows with Connection IDs

Unlike Multipath TCP [2] and Multipath QUIC [5], Multiflow QUIC breaks the symmetry assumption be-

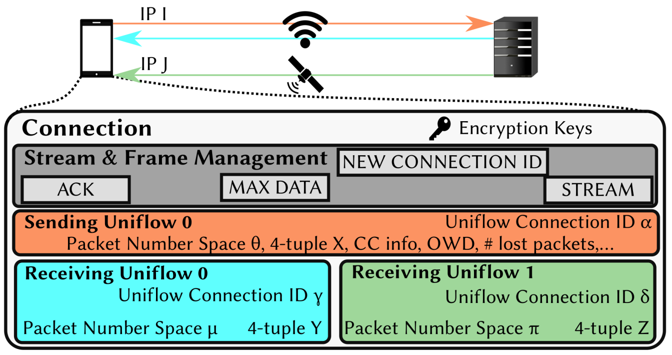

Fig. 1. State of a Multiflow QUIC connection from the client's perspective. The client has IP I when using the upper network path, and IP J when using the lower one.

hind the notion of path and leverages asymmetric Connection IDs to integrate unidirectional flows called *uniflows*. Multiflow QUIC associates packets over these uniflows thanks to specific Connection IDs, called Uniflow Connection IDs. We distinguish two kinds of uniflows, as illustrated in Figure 1. On the one hand, the *sending uniflows* can transmit packets to the remote host. A host can deliver a packet over a given sending uniflow by including the corresponding Uniflow Connection ID in the Destination Connection ID field of the header. On the other hand, the *receiving uniflows* collect incoming packets that reach the host using the related Uniflow Connection ID. Hosts internally identify these uniflows using Uniflow IDs.

Notice that this notion of uniflow is seamlessly included in QUIC [3]. Each packets' flow uses a different Connection ID. Each host sees a given 4-tuple (IP_{src}, IP_{dst}, $port_{src}$, $port_{dst}$) which is not necessarily the mirror of the peer's one since middleboxes like NAT may modify packets' ports and IP addresses. The uniflows on which the connection starts are called the *initial uniflows*. Hosts identify these initial uniflows using Uniflow ID 0.

B. Proposing Sending Uniflows to the Peer

There is a one-to-one mapping between the receiving uniflows of a host and the sending uniflows of its peer. For instance, consider the client's receiving uniflow with ID 1 shown in Figure 1 and mapped to the Uniflow Connection ID δ. For the server, this corresponds to its sending uniflow with ID 1 and carrying packets with the Destination Connection ID δ. Each host controls its number of receiving uniflows — and so the number of sending uniflows of the peer — it wants to maintain. Once a uniflow is initialized, the host can advertise it to its peer using a uniflow-aware version of the NEW CONNECTION ID frame. This frame indicates to the receiver that this Sending Uniflow ID is now usable and that packets can be sent over it by including the communicated Connection ID in their headers. Typically, new uniflows use different network paths and hence other IP addresses. A host can communicate to its peer its current IP addresses using the ADD ADDR frame and associate it with an Address ID. Figure 2 illustrates how a client can propose a new uniflow to the server. Once both the NEW CONNECTION ID and the ADD ADDR frames have been received, the server decides that packets of its sending uniflow 1 are sent to IP address J. However, the server must first ensure that this new IP address actually reaches the client before starting transmitting data. Hence, the server initiates a path validation over this new sending uniflow by delivering the PATH CHALLENGE frame. Since the client has only one sending uniflow — its initial one — it sends the corresponding PATH RESPONSE frame to the initial sending uniflow. Once the path validation completes, the server can start using the provided sending uniflow to send data to the client using both network paths. Finally, the client acknowledges the received packets by sending an ACK frame for each of its receiving uniflows.

The server must validate the provided remote address to prevent flooding a victim pointed by a malicious client and handle the case of clients advertising addresses blocked by a firewall. This process delays the usage of the new uniflow by one round-trip-time. Yet, the validity of the provided address might be cached to save the latency introduced by the path validation.

During the QUIC connection lifetime, it is possible that hosts observe changes in their IP addresses. For instance, a mobile device can obtain a new IP address by powering up its cellular interface and lose its Wi-Fi one because it goes away of access point reachability. To handle these addresses' dynamic, in addition to the ADD ADDR frame, MFQUIC includes the REMOVE ADDR frame to advertise the loss of a previously communicated address. This frame contains the Address ID of the corresponding address along with a sequence number ordering the events related to that particular Address ID.

A host might also discover new peer's addresses by observing the 4-tuple of incoming packets. A client behind a NAT could send its private addresses in ADD ADDR frames, but those are typically not reachable by the server. Instead, the client can initiate the uplink flow on its desired addresses — for instance by performing path validation — such as the server discovers the corresponding public addresses. The server can then validate the observed addresses.

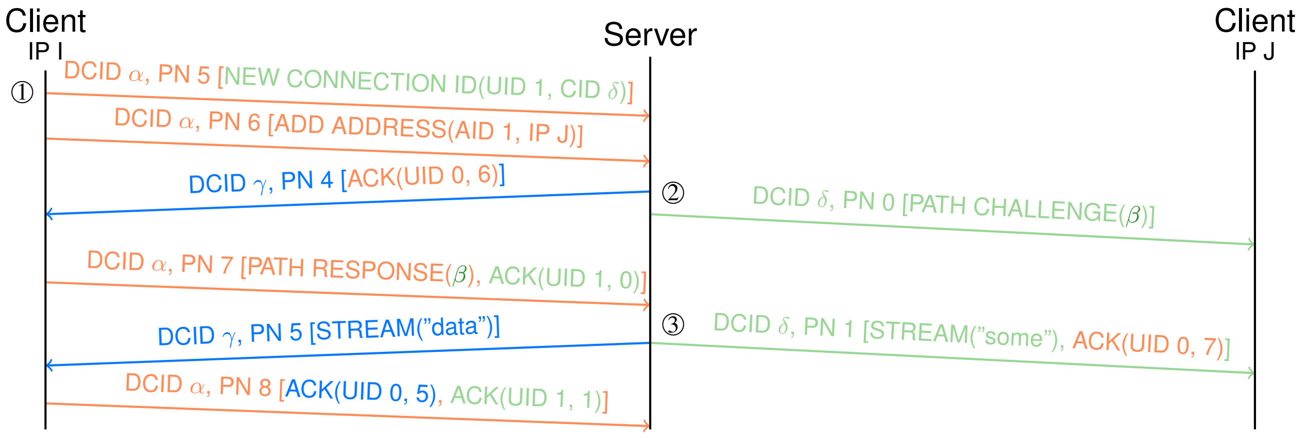
Client
IP I
Server
Client
IP J
①
DCID α, PN 5 [NEW CONNECTION ID(UID 1, CID δ)]
DCID α, PN 6 [ADD ADDRESS(AID 1, IP J)]
DCID γ, PN 4 [ACK(UID 0, 6)]
②
DCID δ, PN 0 [PATH CHALLENGE(β)]
DCID α, PN 7 [PATH RESPONSE(β), ACK(UID 1, 0)]
DCID γ, PN 5 [STREAM("data")]
③
DCID δ, PN 1 [STREAM("some"), ACK(UID 0, 7)]
DCID α, PN 8 [ACK(UID 0, 5), ACK(UID 1, 1)]

Fig. 2. An example of time sequence diagram for a server starting using a new sending uniflow over an established connection in the network scenario shown in Figure 1. The client starts the connection on its IP I and the server always uses the same IP. Only the Destination Connection ID (DCID), the packet number (PN) and the relevant frames are illustrated. Colors show to which uniflow the element relates. AID: Address ID. UID: Uniflow ID. ① Client's Receiving Uniflow 1 ready. ② Server maps Sending Uniflow 1 to destination IP J. ③ Server's Sending Uniflow 1 ready.

C. *Negotiating the Multiflow Extensions*

During the TLS handshake process, QUIC hosts can advertise Transport Parameters carrying internal protocol values. This enables hosts to activate specific features and negotiate the support of an extension. MFQUIC leverages these QUIC transport parameters to negotiate the usage of multiple uniflows. During the handshake, hosts advertise the `max_sending_uniflow_id` QUIC transport parameter to indicate the maximum value of Sending Uniflow ID that they want to support. If both hosts advertise it, the multiflow extensions are enabled on the connection. These values are independent, i.e., hosts may advertise different `max_sending_uniflow_id` values. Because of the mirroring between uniflows, the sending host sets the upper bound of the number of peer's receiving uniflows to the advertised value. Yet, this transport parameter does not impose the peer to provide the desired number of sending uniflows. The receiver keeps the final decision on the number of receiving uniflows it wants to support since it sends the NEW CONNECTION ID frames proposing them.

D. *Ensuring Reliable Data Exchange*

Like Multipath QUIC [5], Multiflow QUIC relies on STREAM frames to carry application data. Because of the separation between packets and frames, hosts can seamlessly spread data over multiple uniflows. The only adaptation to perform is to acknowledge packets on a per-uniflow basis where each uniflow has its own packet sequence number space. MFQUIC adapts the ACK frame to include the host's Receiving Uniflow ID — which is equivalent to the peer's Sending Uniflow ID — it acknowledges. The ACK frame uses the (internal) Uniflow ID instead of the related Connection ID for three reasons. First, the Uniflow ID is shorter than the Uniflow Connection ID. Second, the Connection ID used by packets associated to a uniflow can change over time with the use of NEW CONNECTION ID frames. Third, the Uniflow ID is included in the cryptographic nonce computation to avoid reusing a same nonce over distinct uniflows.

E. *Multiflow-specific Algorithms*

Besides maintaining a congestion window for each sending uniflow, Multiflow QUIC implementations include new algorithms to manage the different sending and receiving uniflows, namely the uniflow manager and the packet scheduler.

a) Uniflow Manager: This algorithm has two responsibilities. First, it decides how many receiving uniflows can be attached to the connection. It therefore determines how many sending uniflows the peer can have. Second, it associates sending uniflows proposed by the peer to a given 4-tuple based on its local view and the received ADD ADDRESS frames. The mapping algorithm can follow similar strategies as proposed in Multipath TCP, such as the `full-mesh` one establishing a sending uniflow between each pair of (local IP, remote IP). This uniflow management is performed by both hosts. Yet, to limit network interferences (NAT, firewalls,...), the server can first wait for the reception of a packet with a given address before using it for its sending uniflows. Notice that the uniflow manager is a generalization of the path manager algorithms used by

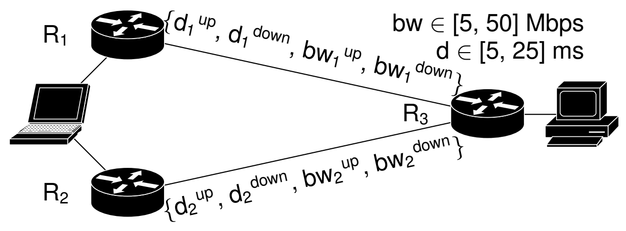
R1
R2
R3
{d1 up, d1 down, bw1 up, bw1 down}
{d2 up, d2 down, bw2 up, bw2 down}
bw ∈ [5, 50] Mbps
d ∈ [5, 25] ms

Fig. 3. Network topology used for asymmetric experiments.

MPTCP and MPQUIC implementations. While any path manager configuration is doable using specific uniflow managers, the reverse is not true.

b) Packet Scheduler: This algorithm has two dimensions as it selects, for each packet, which frames will be sent on which sending uniflow. The packet scheduler notably decides how the application data and ACK frames will be spread over the different sending uniflows. Our MFQUIC prototype uses a lowest-latency-first strategy. To do so, the scheduler takes into account all the available RTT measurements between each pair of (sending uniflow, receiving uniflow). Then, it aggregates these for each sending uniflow by weighting the estimated latency with the number of samples. Each time the packet scheduler needs to send data, it prefers the available sending uniflow with the lowest aggregated latency. However, recall that frames can also carry QUIC control information. Therefore, it is the packet scheduler responsibility to perform the QUIC path validation (using a PATH CHALLENGE frame) for the 4-tuple used by a given sending uniflow. Our scheduler duplicates some control frames on several sending uniflows, such as the MAX DATA and MAX STREAM DATA, to limit possible head-of-line blocking if these important frames are sent on slow or lossy network paths.

IV. Evaluation

We implement Multiflow QUIC as a PQUIC plugin [12] (2800 lines of C code, implements draft-ietf-quic-transport-29) and compare it with our previous Multipath QUIC implementation `mp-quic` [5] (about 5000 lines of Go code change compared to `quic-go`, implements Google QUIC version 39) and the state-of-the-art Multipath TCP (Linux implementation v0.95 [13]). All of these use the lowest latency first packet scheduler and a `full-mesh` flow/path management. Our experimental setup is a lab equipped with Intel Xeon X3440 processors, 16 GB of RAM and 1 Gbps NIC, running Linux kernel 4.19 and configured as shown in Figure 3. The links R_1-R_3 and R_2-R_3 are configured using NETEM to add transmission delays and using HTB to limit their bandwidth.

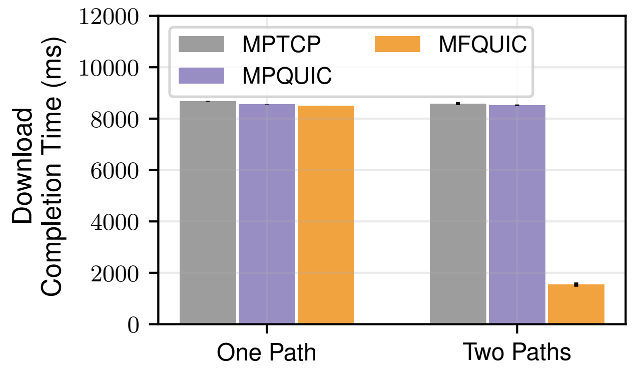

Fig. 4. When the secondary path has a very high asymmetry, MFQUIC benefits from it while both MPTCP and MPQUIC cannot. Error bars show standard deviation over 15 runs.

In this section, we consider the case of a download of a 10 MB file over a single QUIC stream / TCP connection. The client computes the Download Completion Time (DCT) between the GET request and the reception of the last byte of the server response. For each implementation, we perform both one-path experiments over the R_1-R_3 link and two-paths ones using both links. While the two-paths runs use the aforementioned protocols, the one-path ones rely on the corresponding single-path variant of each implementation, i.e., we disable multipath/multiflow features for these one-path experiments. We first focus on a specific network scenario and then explore a large set of parameters.

A. Case Study of A Highly Asymmetric Network Path

To highlight the benefits of Multiflow QUIC, consider the following scenario. The primary network path exhibits symmetric characteristics with 10 Mbps and 20 ms RTT. The secondary one has good downlink performance (50 Mbps, 10 ms one-way delay) but very poor uplink one (0.01 Mbps, 1 s one-way delay). Such setup is close to a satellite network path. Figure 4 shows that when using only the primary network path, both single-path QUIC implementations and TCP complete the download in about 8.5 s. Since all protocols share the same (symmetric) network path, this result is expected. However, the usage of the highly asymmetric network path makes the MFQUIC transfer about 5.5 times quicker, while neither Multipath TCP nor MPQUIC can take advantage of it. This difference resides in the ACK frame scheduling strategy. While MFQUIC can send ACK frames on any sending uniflow (typically the one using the primary network path), MPQUIC carries ACK frames on the same path and Multipath TCP needs to keep acknowledgments at subflow-level. Due to the poor path

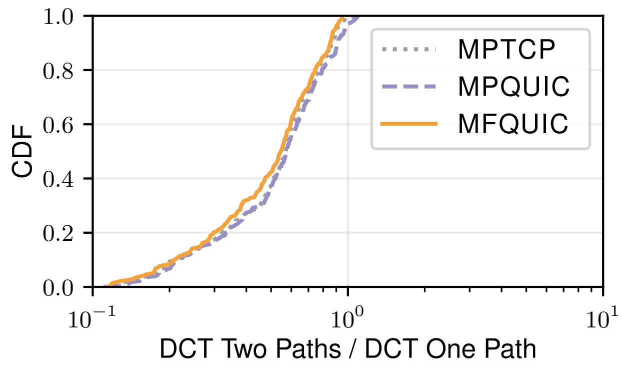

Fig. 5. In our parameter space, MFQUIC slightly benefits more from the addition of a second asymmetric network path than both MPTCP and MPQUIC.

uplink performance, both Multipath TCP and MPQUIC senders receive acknowledgments with a delay of about 1 second. The congestion window of the secondary path increases slowly and only a few data bytes (about 90 KB out of 10 MB for MPQUIC) are sent on it. In comparison, MFQUIC carries most of the data bytes over the secondary path (about 8.4 MB out of 10 MB). Notice that both MFQUIC and MPQUIC prevent head-of-line blocking by duplicating MAX DATA and MAX STREAM DATA frames on both network paths [5].

B. Exploring Asymmetric Network Use Cases

While beneficial in the previous case, one could question the usefulness of uniflows in asymmetric networks in general, including their performance impact. To address this, we now cover a large range of link characteristics by applying an experimental design approach [14] and defining the parameters' ranges as shown in Figure 3. We use the WSP algorithm [15] to broadly sample this parameter space into 139 points. Notice that the uplink one-way-delay d^{up} and the downlink one d^{down} are two different parameters. The same applies for the uplink bandwidth bw^{up} and the downlink one bw^{down}. Since our topology features two network paths, we explore an 8-dimension parameter space. For each parameter configuration, we vary the initial network path used, resulting in 278 different network scenarios. Each scenario is run 9 times and the median run is reported.

Figure 5 shows the speedup achieved by Multipath TCP and both QUIC flavors when using a second network link. Each point of the cumulative distribution function represents one of the 278 network scenarios. Multipath TCP and both QUIC implementations achieve lower DCT when using the two links. In our considered experiments, Multiflow QUIC (resp. Multipath QUIC, Multipath TCP) achieves a median speedup of 83.5% (resp. 73%, 75%). By computing the DCT ratios between the implementation themselves, Figure 6 shows that all implementations in single-path mode (i.e., plain TCP and both single-path QUIC implementations) achieve similar performance. This means that comparing them directly, Multiflow QUIC is in general slightly faster than Multipath QUIC and Multipath TCP.

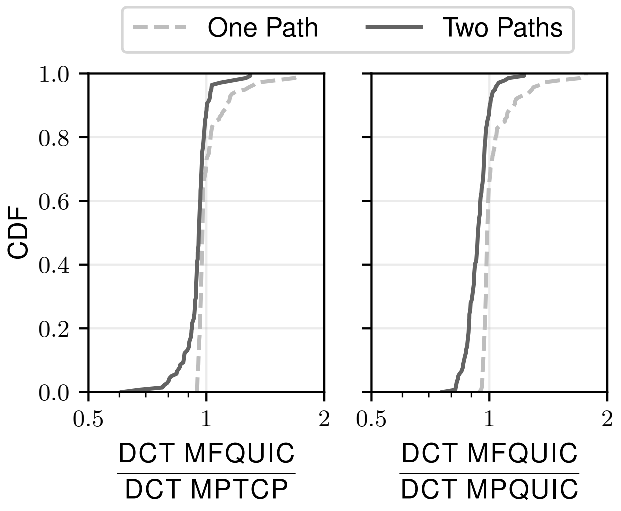

Fig. 6. Comparing the performance of the Multiflow QUIC's implementation to the Multipath TCP's one (left sub-figure) and the Multipath QUIC's one (right sub-figure) on each network scenario.

Again, the scheduling strategy of ACK frames explains these results. Multipath TCP and Multipath QUIC always acknowledge packets on the same path. Multiflow QUIC gets rid of this constraint and sends ACK frames on the preferred sending uniflow, taking advantage of the best performing one. This enables Multiflow QUIC senders to increase their sending uniflows' congestion window quicker than with Multipath TCP and Multipath QUIC, especially in the cases where there is a strong asymmetry between uplink and downlink.

V. Conclusion

With TCP, reliable transport protocols used to consider bidirectional paths and stuck with them. Multipath TCP and Multipath QUIC keep this constraint that limits their performance and their applicability in some scenarios. This paper broke this assumption and proposed to be more generic by considering unidirectional flows instead. We then designed Multiflow QUIC to address scenarios where multiple (possibly asymmetric) network paths are available. Our evaluation revealed that MFQUIC can take advantage of unidirectional network paths while MPTCP and MPQUIC cannot. We also showed that scheduling packets on a uniflow basis slightly increases transfer performance in asymmetric network scenarios. Compared to existing multipath transport protocols, MFQUIC opens new use cases and perspectives of better taking advantage of network paths. For instance, with MFQUIC it becomes possible to create a hybrid satellite service where clients combine a high capacity downlink satellite link with a lower speed by bidirectional terrestrial network.

ARTIFACTS

Our studied implementations are freely available: Multipath QUIC at https://github.com/qdeconinck/mp-quic and Multiflow QUIC at https://github.com/p-quic/pquic. Our measurement scripts and data are available at https://multipath-quic.org/commag21.

REFERENCES

[1] O. Bonaventure and S. Seo, "Multipath TCP Deployments," in *IETF Journal*, Nov. 2016, vol. 12, no. 2, pp. 24–27.

[2] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020, accessed on 2020-09-30. [Online]. Available: https://rfc-editor.org/rfc/rfc8684.txt

[3] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-34, Jan. 2021, work in Progress, Accessed on 2021-01-21. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34

[4] A. Langley *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*. Los Angeles, CA, USA: ACM Press.

[5] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and Evaluation," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies - CoNEXT '17*. Incheon, Republic of Korea: ACM Press, December 2017, pp. 160–166.

[6] F. C. Commission, "Measuring fixed broadband - ninth report," Aug. 2020, accessed on 2020-11-30. [Online]. Available: https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-ninth-report

[7] B. Augustin, T. Friedman, and R. Teixeira, "Measuring multipath routing in the internet," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 830–840, 2010.

[8] H. Balakrishnan and V. N. Padmanabhan, "How network asymmetry affects TCP," *IEEE Communications Magazine*, vol. 39, no. 4, pp. 60–67, 2001.

[9] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi, "A survey on recent advances in transport layer protocols," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3584–3608, 2019.

[10] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is It Still Possible to Extend TCP?" in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference - IMC '11*. Berlin, Germany: ACM Press, 2011, pp. 181–194.

[11] Q. De Coninck and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)," Internet Engineering Task Force, Internet-Draft draft-deconinck-quic-multipath-06, Nov. 2020, work in Progress, accessed on 2021-01-25. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-deconinck-quic-multipath-06

[12] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing QUIC," in *Proceedings of the ACM Special Interest Group on Data Communication - SIGCOMM '19*. Beijing, China: ACM Press, 2019, pp. 59–74.

[13] C. Paasch, S. Barre *et al.*, "Multipath TCP in the linux kernel," 2019, http://www.multipath-tcp.org, accessed on 2020-11-23.

[14] R. A. Fisher, "The design of experiments." 1935.

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

Quentin De Coninck (quentin.deconinck@uclouvain.be, https://qdeconinck.github.io) received his Ph.D. from UCLouvain, Belgium in 2020 under the supervision of Prof. Olivier Bonaventure. He is now a post-doctoral researcher in the same institution. His research interests include Internet protocol design and low-level system architecture.

Olivier Bonaventure is Professor at UCLouvain where he leads the IP Networking Lab. He has actively contributed to the design, implementation and deployment of several Internet protocols including LISP, Multipath TCP, IPv6 Segment Routing, ... He served as Editor for SIGCOMM's Computer Communication Review. He wrote the *Computer Networking: Principles, Protocols and Practice* open-source ebook that is used by various universities. He co-founded the Tessares spinoff that deploys Hybrid Access Networks using Multipath TCP.