MultipathTester: Comparing MPTCP and MPQUIC in Mobile Environments

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

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

***Abstract*—With the adoption of Multipath TCP by Apple for its iPhones, there is a growing interest in using multipath transport to seamlessly support different network interfaces on mobile devices. In parallel, the IETF is actively developing the QUIC protocol which could replace TCP for some types of applications and multipath extensions for QUIC have already been proposed. We design and implement MultipathTester, an iOS application that enables researchers to compare the performance of different multipath protocols on popular smartphones. Specifically, our application compares the performance of iOS Multipath TCP with Multipath QUIC with various traffic types and conditions.**

I. Introduction

Many mobile devices such as smartphones, tablets or connected cars are multihomed devices. They can be attached to different networks such as cellular and Wi-Fi simultaneously. This multi-homing capability enables mobile devices to switch from one wireless network to another one without user intervention. Unfortunately, TCP cannot spread connection data over multiple network interfaces, as each connection is bound to the 4-tuple (IP_{src}, $port_{src}$, IP_{dst}, $port_{dst}$).

Several solutions have been proposed to enable mobile devices to switch from one network to another or use different networks simultaneously, but only one of them, Multipath TCP [1], has been commercially deployed, especially with mobile devices. In a nutshell, Multipath TCP is a TCP extension that divides a connection into several TCP subflows. Each TCP subflow carries a fraction of the TCP and the receiver reorders the data received over the different subflows to deliver them in sequence. Another benefit of Multipath TCP is that it if one of these subflows fails, Multipath TCP can seamlessly shift its connection on another subflow without involving application intervention. This seamless handover capability convinced Apple in 2013 to integrate Multipath TCP to support the Siri voice-activated application, making iPhones the largest deployment of Multipath TCP [2]. For four years, Siri was the only application on iPhone that took advantage of such multipath feature. But since September 2017, any iOS11 application can now request the usage of Multipath TCP. As of today, 97 % of the iPhones have been upgraded to iOS11 or greater [3]. Several Android smartphones also support Multipath TCP to seamlessly use Wi-Fi and LTE [2], [4], but they are currently limited to specific countries.

Quentin De Coninck is a F.R.S.-FNRS Research Fellow.

978-3-903176-17-1 / © 2019 IFIP

Although smooth handovers are one of the key benefits of Multipath TCP, its performance under those conditions have not yet been analyzed in details. The first study of those handovers [5] confirmed that the Linux implementation of Multipath TCP could provide smooth handovers but it did not analyze various networks environments. Most of the scientific literature on Multipath TCP focused on its bandwidth aggregation capabilities [6]–[8], despite the importance of smooth handovers for mobile applications.

Recently, the networking community and the IETF have worked on the design and implementation of the QUIC [9] protocol aiming at providing the services of TCP, TLS and HTTP atop UDP. QUIC is being finalized within the IETF [10] and its first RFC is expected in July. This initial design will support a single path for each connection. Once the initial specification has been approved, the QUIC working group will probably look at adding multipath support to the QUIC protocol. An initial design for Multipath QUIC [11], [12] has already been proposed. Similarly to Multipath TCP, Multipath QUIC allows the simultaneous usage of multiple network paths for a given connection. However, Multipath QUIC was only evaluated in emulated environments [11] and its performance in real networks has not been studied yet.

To encourage the evaluation of (Multipath) TCP and (Multipath) QUIC, we design and implement **MultipathTester** [13], an iOS application that tests how these different protocols behave under various conditions. It provides two experimentation modes. The first one generates different traffic patterns ranging from bulk transfer to delay-sensitive request-response and observes how Multipath TCP and (Multipath) QUIC operates within stable network conditions. The second one requires the user to move until it causes a network handover from Wi-Fi to cellular networks and observes how multipath protocols handle such changing network conditions. To our knowledge, this is the first Internet-wide comparison of Multipath TCP and Multipath QUIC. We believe our open measurement platform would enable the community to study new QUIC extensions in such mobile environments.

This paper is organized as follows. Section II presents the architecture of our measurement platform. Then, Section III provides initial results about the usage of QUIC in real networks with stable conditions. Section IV summarizes our first results on how Multipath TCP and Multipath QUIC handle handovers. Finally, Section V concludes this work.

II. Design of MultipathTester

In this section, we introduce MultipathTester, an iOS application aiming to evaluate the performance of multipath protocols. Our framework enables researchers to observe how multipath protocols behave under our different test modes we present first. We then elaborate on our measurement infrastructure and implementation details. We finally provide a few statistics about the usage of our application.

A. Test Modes

MultipathTester relies on active measurements performed by voluntary users. For this purpose, at the first run and before performing any measurement, the application provides a consent form describing its research purpose to the user. Once agreed, the user can explicitly start generating active network traffics. Two kinds of experiments are available. First, the stable network mode benchmarks the connectivity using different traffic patterns. Second, the mobile mode studies the impact of network handovers on multipath protocols.

1) Stable Network Mode: During a stable network test, the user is expected to stay at the same place so that the smartphone remains attached to the same network during the entire test. In these stable conditions, we benchmark the access point(s) to check if the studied transport protocols behaves correctly in these networks. MultipathTester monitors the network connectivity during the test to detect user moves. It relies on the `Reachability` API [14] to keep the list of available network interfaces from the device point of view. If during its run, the availability of one network interface changes (e.g., Wi-FI being declared as lost by the device, cellular just getting Internet connectivity,...), the test is interrupted and classified as invalid.

Different traffic patterns can be used with this mode. Stable network tests launch these traffic patterns sequentially, one transport protocol at a time. The order of the runs are randomized to avoid possible traffic correlation. MultipathTester provides a common interface to define traffic patterns. We currently explore four simple patterns, yet easy to explain: *ping*, *bulk*, *iperf* and *interactive*.

Ping. This test simply sends a stream of five HTTP GET requests for a 10-byte file, and computes the median delay. It is mainly used to check the connectivity and to select the server with which further experiments will be performed. Our experiment servers are currently located on different continents: Europe (France), North America (Canada) and Asia (Japan).

Bulk. This test performs an HTTP GET request for a 10-MB file, and records the download time.

IPerf. This test generates traffic similar to the `iperf` tool. The client sends new data as fast as possible for a few seconds. We currently use it to estimate the uplink bandwidth. The downlink support is part of our future work.

Interactive. This traffic simulates a user interacting with a voice-activated application such as Siri while listening to an Internet radio. To achieve this, the traffic pattern follows a bi-directional request/response fashion as shown in Figure 1.

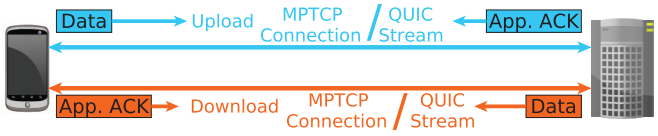


Fig. 1: Interactive traffic.

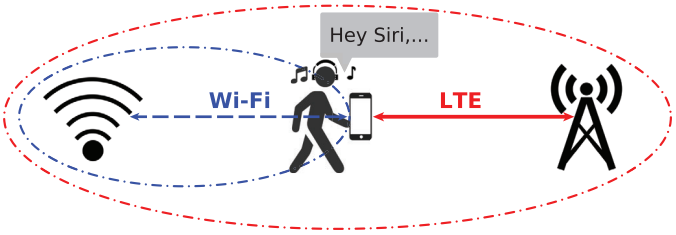


Fig. 2: Mobility enables handover tests.

Every 100 ms, both the client and the server send a 2KB request to the peer that replies with a 750 bytes response. On both data streams, the receiving host returns a short application-level acknowledgment that confirms the reception of each 2KB chunk. The sending host then computes the delay between the request and the corresponding acknowledgment. With such low-volume exchanges, we do not expect any interference with the receive nor the congestion windows. Notice that we had to implement this traffic pattern differently for Multipath TCP and Multipath QUIC. With (Multipath) TCP we use two independent connections to prevent head-of-line blocking where a lost response blocks the delivery of the next request. With (Multipath) QUIC we use a single connection that carries two independent data streams.

2) Mobile Mode: Our mobile tests focus on the situation presented in Fig. 2. A user is initially connected to both Wi-Fi and cellular networks while sending and receiving data simultaneously using the *interactive* traffic pattern. Then, the device moves away from the Wi-Fi access point. After some time, as the Wi-Fi connectivity is fading, multipath transport protocols switch to the cellular one. This network handover situation is one of the motivating cases for supporting multipath in the transport layer, as they can migrate connections from one network to another one without notifying the application. We simultaneously evaluate multipath protocols, meaning they compete for the network interfaces at the same time. However, due to the low total generated traffic volume (55 KB/s uplink and 55 KB/s downlink), we believe that the impact on the observed delays should be negligible. Furthermore, running them simultaneously enables us to evaluate protocols within the same singular network conditions due to the device mobility. We expect the impact of the handover to be as low as possible on the applications, especially when they are latency-sensitive. To encourage users to perform this test, we present it as a Wi-Fi reachability estimator where the smartphone computes the range of the Wi-Fi access point. The test completes when either the operating system tears down the Wi-Fi network, or the SSID of the Wi-Fi network changes.

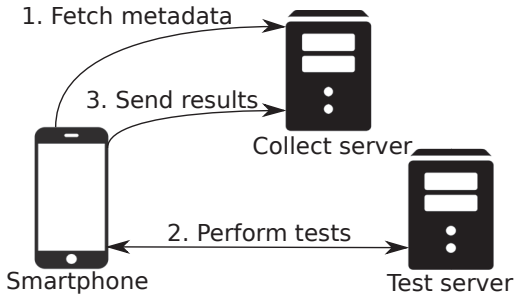


Fig. 3: The infrastructure used by MultipathTester.

B. Measurement Infrastructure

Our measurement infrastructure involves three different nodes, as shown in Figure 3. On one side, there is the smartphone running MultipathTester. On the other side, we use two different servers. The test servers are contacted by the smartphone to perform the experiments. We currently use three test servers located in Europe, Asia and America. Second, the collect server gathers the measurement results. Each user-triggered measurement is carried out as follows. First, the smartphone contacts the collect server to fetch metadata, such as the URL of available test servers and the list of experiments to launch. Then, once the user requests it, the application interacts with the closest test server to perform experiments. At the end of the test runs, the smartphone sends the test results to the collect server. These results include primary traffic-specific metrics (delays for *interactive*, download completion time and file fetched for *bulk*,...), device and network information (name and type of the network accesses, version of the application,...) and dumps of the transport protocol states. These dumps are periodically collected. For (Multipath) TCP, we rely on both the `TCP_INFO` and the `ioctl SIOCGCONNINFO` interfaces. Our (Multipath QUIC) implementation itself logs its internal variables using a dedicated thread in a file.

On both smartphone and test servers, we use our `mp-quic` implementation to serve (Multipath) QUIC [11]. This implementation is based on a old GQUIC version using a different network format than IETF QUIC. However, except for the QUIC connectivity, we do not expect much difference with IETF QUIC in terms of (multipath) performance. Test servers use the Multipath TCP implementation in the Linux kernel 4.14 [15] with default parameters (default low-RTT scheduler, fullmesh path manager and the OLIA congestion control scheme [16]). The smartphones use the native implementation of (Multipath) TCP provided by the Darwin kernel [17].

On iOS, applications can explicitly request the usage of Multipath TCP using the iOS API. Apple provides three modes of operation for Multipath TCP, each with different objectives: *handover*, *aggregate* and *interactive*. The *handover* mode aims to provide seamless handover from Wi-Fi to cellular networks for long-lived or persistent connections. The *aggregate* mode uses all network connectivities to increase the throughput of the connection. The *interactive* mode attempts to use the lowest-latency connectivity and is advised for latency-sensitive, low-volume connections. Nonetheless, the ability to use the LTE network while the Wi-Fi one might still be available raises concerns about cellular data consumption. Users typically expect the device to use the Wi-Fi network when available, even if it provides lower throughput and/or larger latency than the cellular one. This is why Apple restricts the *aggregate* mode to developer phones only. Since we want our application to be as accessible as possible, we do not support the *aggregate* mode.

When requested, the smartphone uses the *interactive* Multipath TCP mode. We focus on this mode rather than the *handover* one as it is advised for latency-sensitive applications, which matches our *interactive* traffic. We inferred its operations based on its source code [17]. The interactive mode prioritizes the Wi-Fi network over the cellular one, marking the latter one as a backup subflow. The iOS packet scheduler sends data only on the Wi-Fi subflow, unless one of the following conditions occurs.

1) The smoothed RTT of the Wi-Fi subflow is above a threshold initially set to 600 ms, while the cellular path is not over this threshold;
2) The Wi-Fi path is experiencing RTO, i.e., the timer has fired and no acknowledgment was received since that event, and the phone wants to push new data;
3) The Wi-Fi RTO value is over a threshold initially set to 1500 ms, while the cellular path is not.

Notice that the threshold values can be decreased by the Apple's WifiAssist application when it considers the Wi-Fi network as "bad". However, this system is closed-source, making it difficult to understand its operation.

To avoid being unfair with regard to Multipath TCP, we configured the Multipath QUIC scheduler such as it also advertise all cellular paths as backup ones. This prevents QUIC from using the cellular path directly. If the smartphone notices RTO on the Wi-Fi path or some data being in-flight for more than 600 ms, it starts using the cellular path.

MultipathTester also integrates basic visualizations on most interesting variables for the user, either while running the test or when looking at results. The application contains ∼9250 lines of code (without comments), whose ∼ 7000 are Swift, 1000 are Objective-C and ∼1250 are Go code.

C. Usage Statistics

Since the first public release of MultipathTester on March 8th, 2018 until April 30th, 2019, we collected 1098 test runs coming from 264 unique devices. 43% of the runs are mobile tests. The distribution of test loads between Europe, Asia and America is 65%, 17% and 18%, respectively. MultipathTester has been used in 72 different mobile carriers and 288 different Wi-Fi SSIDs.

III. Stable Network Runs

In this section, we briefly describe some interesting results obtained during stable network tests. We first provide single-path findings and then expand on multipath ones.

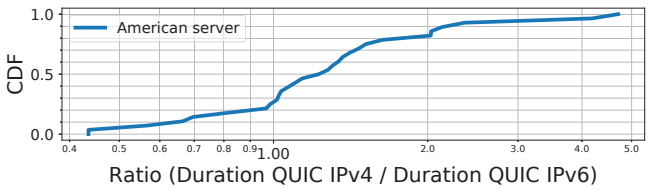


Fig. 4: In America, we observe that IPv6 offers better results than IPv4, probably due to NAT64.

IPv6 Connectivity. Proportionally, the American server observes the largest proportion of IPv6 compatible hosts, with 65% of smartphones having an IPv6 address. In comparison, the European one only observes 43% of the devices with IPv6 addresses, and on the Asian one, this number drops to 29%. However, we also observe that having an IPv6 address does not guarantee QUIC connectivity using IPv6. On the European server, if we select the devices having both IPv4 and IPv6 addresses, we observe a QUIC connectivity success rate of 89% using IPv4, but this rate decreases to 58% over IPv6. When digging into the tests where the IPv4 QUIC handshake succeeded but not the IPv6 one, we notice two kinds of error equally balanced. The first one is simply the QUIC handshake in IPv6 that timeouts. Most of the times, this happens when IPv6 is provided by the Wi-Fi network. Indeed, this issue arose in 12 different Wi-Fi networks, while only 2 different cellular ones suffered from such timeouts. The second error cause is the QUIC client that encounters a "no route to host" error while trying to send the packet to an IPv6 address. This typically occurs when the smartphone selects the Wi-Fi network as its default interface while it only provides IPv4 connectivity, even though an IPv6 address is present at the cellular interface. Such routing issue is probably due to a bad interaction between iOS and the QUIC implementation.

Performance of QUIC using IPv4 vs. IPv6. When QUIC is usable over both IPv4 and IPv6, we do not observe much difference in terms of performance using the different traffic patterns on the European server. However, on the American one, we see better results with IPv6 than with IPv4. For instance, Figure 4 provides the ratio of the download completion times of a 10MB file between QUIC IPv4 and QUIC IPv6. In 75% of the measurements, using IPv6 leads to shorter completion times than using IPv4. Using the same test set and looking at the *interactive* traffic pattern, we observe lower maximum experienced delays with QUIC IPv6 than QUIC IPv4. This might be related to the high deployment of IPv6 in American networks, where the IPv4 connectivity is provided by NAT64. Other studies [18]have shown that IPv6 was faster than IPv4 in mobile networks.

Usage of QUIC on unofficial ports. QUIC usually runs on port 443, but some middleboxes might expect other protocols such as DTLS on this port and could interfere with QUIC [9]. To detect the presence of such middleboxes, we also run our *ping* traffic with QUIC on the non-standard port 6121 to observe if it experiences connectivity issues. Globally, we do not observe much difference using port 6121 and the standard 443 one.

Reason	RTT Thres.	Under RTO	RTO Thres.	Other
Test (%)	2.2%	67.0%	11.0%	19.8%

TABLE I: Multipath TCP reason to start using the cellular.

Performance of MPQUIC vs. MPTCP. When the smartphone can use both Multipath TCP and Multipath QUIC, we observe similar performance for each protocol with our different traffic patterns. This is expected as we applied the same scheduling strategy preferring the Wi-Fi network on both protocols. However, especially with the *iperf* traffic pattern, we notice that when the network offers a large upload bandwidth, i.e., over 50 Mbps, Multipath TCP achieves a much higher throughput than Multipath QUIC. This is probably related to the implementation overhead. Multipath TCP in the Darwin kernel is much more optimized than the `gomobile` framework making the link between Swift code and the `mp-quic` implementation written in Go.

IV. Mobile Experiments

In this section, we focus on the performance of multipath protocols when the user moves. MultipathTester currently uses only the *interactive* traffic pattern while the user moves. This section is split into two parts. We first focus on the *interactive* mode of iOS Multipath TCP and evaluate whether it achieves its low-latency goal. Then we provide a first comparison of the performance of Multipath QUIC with Multipath TCP.

A. Multipath TCP and its Interactive Mode

We consider here a dataset of 231 experiments performed between April 23rd, 2018 and April 30th, 2019. Our mobile dataset includes 44 distinct cellular networks and 84 different Wi-Fi ones.

Multipath TCP often waits for Wi-Fi RTO before using cellular. The Multipath TCP interactive mode follows the algorithm described in Sect. II-B to decide when the smartphone should start using the cellular backup path. Thanks to the periodic collection of Multipath TCP internal state, we can infer which condition triggered the usage of the cellular path by the smartphone. Table I shows that two-third of the tests started to use the cellular because new data arrived while the Wi-Fi subflow was experiencing an RTO. This might be related to our `interactive` traffic pattern that generates data every 100 ms. In comparison, handovers caused by high smoothed RTTs are rare. This might be related to the high initial threshold of 600 ms. Notice that the reason for 20% of the cellular switches cannot be determined using the three first conditions. We suspect that WifiAssist has declared the Wi-Fi network as "bad" and decreased the RTT and RTO thresholds. However, these thresholds are not exposed by the Darwin kernel, making impossible to confirm this hypothesis.

A Multipath TCP handover is not an abrupt process. As the smartphone moves away from the Wi-Fi access point, its performance will eventually decrease, leading to a network

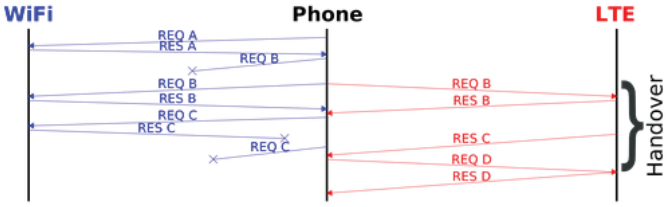


Fig. 5: Example of a possible network handover.

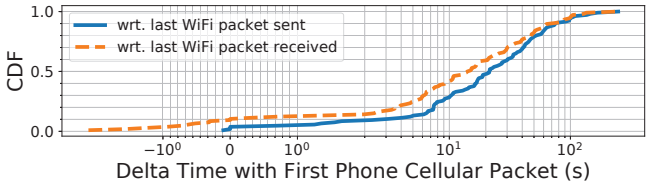


Fig. 6: Duration of the WiFi to cellular handover.

handover to the cellular network. However, this switch is not necessarily instantaneous. Consider the situation shown in Fig. 5. The connection starts over the Wi-Fi network, and after some time, it experiences retransmissions due to weaker signal. The phone then decides to use the cellular path to retransmit the lost request as it experienced an RTO on Wi-Fi, while still retransmitting the packet over the Wi-Fi path too. These retransmissions might eventually succeed, leading to reusing the Wi-Fi path. Therefore, there is a time interval during which both Wi-Fi and cellular networks are still functional and being used by the connection. We call this transient state the *handover duration*.

The ability of using both networks concurrently enables the smooth handover. To quantify its duration, we measure the delay between the first data packet sent on the cellular network and the last activity observed on the Wi-Fi network. The end of the Wi-Fi liveliness can be measured as either the transmission time of the last packet (data or TCP acknowledgment) sent by the phone or the reception time of the last packet received. The advantage of the second metric is that it provides a better view about the actual availability of the Wi-Fi network. Figure 6 shows when smartphones move, they simultaneously use the Wi-Fi and cellular networks. Indeed, only 10% of the test runs experiences an abrupt switch from Wi-Fi to the cellular network, i.e., the Wi-Fi stopped working before the smartphone started to use the cellular path. This corresponds to the negative values in Figure 6. On the other hand, 58% of the experiments observe a handover duration of at least 10 seconds. This illustrates that in mobile scenarios, the network handover is not an abrupt process.

Multipath TCP network handovers can affect the application. The main objective of the interactive mode of Multipath TCP is to enable the application to automatically use the lowest-latency interface while keeping the usage of the cellular one as low as possible. To this end, we focus on the maximum delay observed by the sending host for both upload and download streams of the *interactive* traffic pattern, i.e.,

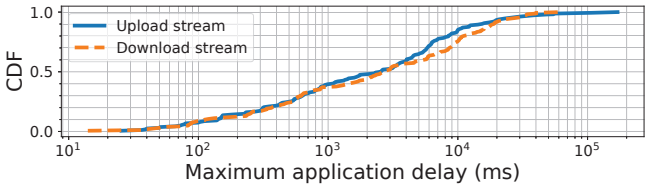


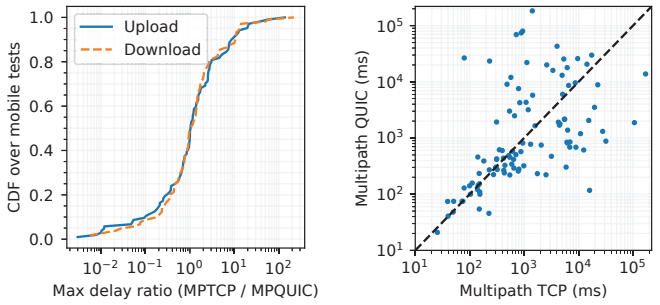
Fig. 7: Multipath TCP maximum observed application delays.

two maximum delays are collected per test and per protocol. Despite the interactive mode, Figure 7 shows that more than 60% of the test cases have a maximum application delay that is longer than a second for both data streams. To exemplify our results, imagine that the download stream represents an Internet radio while the upload one simulates a voice-activated application such as Siri. With the median value at 2.6 s, this means that the user would have experienced a music stall in 50% of the experiments if the playback buffer was not at least 2.5-second long. Similarly, with 70th percentile of 6 s, the Siri application would have been unresponsive for at least 6 s, possibly raising a connectivity error to the user. From our experiments, we observe that upload connections tend to have lower application delays than download ones, especially between percentiles 50th and 90th. This is likely because the phone has a better view of the network, as it is easier for it to detect a weak Wi-Fi than the server. As in the upload connection, the phone is sending the data, it can adapt its packet scheduler thanks to its local information. On the opposite, the server needs to rely on retransmissions. We also observe a tail reaching hundreds of seconds. This is due to the iOS implementation of Multipath TCP that does not support the Multipath TCP `ADD_ADDRESS` option [1] for privacy reasons. Our test servers have both IPv4 and IPv6 addresses. With the happy eyeballs process, if the connection starts on the Wi-Fi using IPv6 while the cellular is IPv4-only, the transfer remains stuck on the Wi-Fi. Note that this issue does not occur in the converse situation (IPv4 Wi-Fi with IPv6-only cellular), as the iPhone includes a NAT64 daemon.

B. Comparison with Multipath QUIC

With the built-in support of Multipath TCP in iOS and Multipath QUIC provided by our application, MultipathTester can compare how both protocols handle network handovers when the user moves. Here, we study a subset of the previously described dataset where both protocols were usable. This dataset contains 104 experiments, involving 40 cellular networks and 61 Wi-Fi ones.

To compare their performance, we consider the maximum delay experienced by each of the protocol over the same run. We then compute for each mobile test the ratio between the maximum delay of Multipath TCP and Multipath QUIC. Figure 8a shows that when Multipath QUIC uses the same scheduling strategy as Multipath TCP, we do not observe a clear trend in favor of one protocol over the other. Each of them tends to start using the cellular interface at the same



(a) Ratio of max delays between MPTCP and MPQUIC.

(b) Upload stream max delays. Each point represents a test.

Fig. 8: The performance of the two protocols mainly depend on the network conditions.

time. In addition, Fig. 8b indicates that for a same test run, we can observe very different experienced maximum delays between Multipath TCP and Multipath QUIC. This result is both surprising and encouraging. Although Multipath TCP is included in iOS11, it does not seem to detect handovers better than our application. This indicates that smartphone applications that will include Multipath QUIC in the future could reach similar handover efficiencies as Multipath TCP.

As Multipath TCP is implemented in the kernel, applications cannot easily tune it according to their needs. As explained earlier, Apple does not support the utilization of both Wi-Fi and cellular for bulk transfers with Multipath TCP. However, this limitation does not exist for UDP. An iOS11 application that uses Multipath QUIC could thus use both Wi-Fi and cellular simultaneously. It could also easily use another packet scheduler such as those that were proposed for Multipath TCP or a different congestion control scheme. MultipathTester could thus enable a wide range of multipath experiments.

V. Conclusion

In this paper, we presented MultipathTester, an iOS application acting as a measurement platform to compare the performance of Multipath TCP and (Multipath) QUIC. Our application enables users to perform two kinds of experiments. First, the stationary mode evaluates how the studied protocols behave in real Wi-Fi and cellular networks under stable network conditions. We notably noticed some connectivity issues with QUIC using IPv6, but in America we observed better performance using IPv6 rather than IPv4. Second, the mobile mode evaluates the impact of network handovers on a delay-sensitive application running atop Multipath TCP and Multipath QUIC. We learned that the Wi-Fi to cellular handover process often takes more than a second and that with similar scheduling strategies, Multipath TCP and Multipath QUIC achieves similar results.

While the experimentation with Multipath TCP is limited to the API exposed by iOS, there is no restriction with Multipath QUIC. We expect that our platform could be used to evaluate new QUIC extensions. For instance, a possible next step could be to include a Forward Erasure Correction extension to QUIC [19] and assess how applications could leverage it when experiencing network handovers. Once Multipath TCP will be integrated in the mainstream Linux kernel, we could also extend MultipathTester to the Android platform and explore how the smartphones' vendors tune their multipath algorithms.

Artifacts. Our iOS application, the traffic pattern implementations in Go and the collection server are available at https://github.com/multipathtester.

Acknowledgements

We thank the anonymous reviewers and our shepherd Brian Trammell for his valuable comments.

References

[1] A. Ford *et al.*, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Internet Engineering Task Force, January 2013.
[2] O. Bonaventure and S. Seo, "Multipath tcp deployments," *IETF Journal*, vol. 12, no. 2, pp. 24–27, 2016.
[3] D. Smith, "iOS Version Stats," May 2019. [Online]. Available: https://david-smith.org/iosversionstats/
[4] Z. Cao, "Multi-path transport deployment on smartphone apps," March 2019, presentation at IETF104, MPTCP working group.
[5] C. Paasch *et al.*, "Exploring mobile/wifi handover with multipath tcp," in *CellNet'12*. ACM, 2012, pp. 31–36.
[6] S. Deng *et al.*, "Wifi, lte, or both?: Measuring multi-homed wireless internet performance," in *IMC'14*. ACM, 2014, pp. 181–194.
[7] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen, "An in-depth understanding of multipath tcp on mobile devices: measurement and system design," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 189–201.
[8] Y.-C. Chen *et al.*, "A measurement-based study of multipath tcp performance over wireless networks," in *IMC'13*. ACM, 2013, pp. 455–468.
[9] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 183–196.
[10] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-20
[11] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017, pp. 160–166.
[12] Q. D. Coninck and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)," Internet Engineering Task Force, Internet-Draft draft-deconinck-quic-multipath-02, Mar. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-deconinck-quic-multipath-02
[13] Q. D. Coninck, "Multipathtester," 2019. [Online]. Available: https://itunes.apple.com/us/app/multipathtester/id1351286809?mt=8
[14] Apple, "Reachability," 2016. [Online]. Available: https://developer.apple.com/library/archive/samplecode/Reachability/Introduction/Intro.html
[15] C. Paasch, S. Barre *et al.*, "Multipath tcp in the linux kernel," 2009-2018, http://www.multipath-tcp.org.
[16] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "Mptcp is not pareto-optimal: performance issues and a possible solution," *IEEE/ACM Transactions on Networking (ToN)*, vol. 21, no. 5, pp. 1651–1665, 2013.
[17] Apple, "Xnu kernel source code," 2018. [Online]. Available: https://opensource.apple.com/source/xnu/
[18] A. Dhamdhere, M. Luckie, B. Huffaker, A. Elmokashfi, E. Aben *et al.*, "Measuring the deployment of ipv6: topology, routing and performance," in *Proceedings of the 2012 Internet Measurement Conference*. ACM, 2012, pp. 537–550.
[19] F. Michel, Q. De Coninck, and O. Bonaventure, "Quic-fec: Bringing the benefits of forward erasure correction to quic," in *2019 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2019.