On the Experimental Assessment of QUIC and Congestion Control Schemes in Cellular Networks

Mohamed Moulay[‡], Fernando Díez Muñoz[*†], Vincenzo Mancuso[*]
[*]IMDEA Networks Institute, Madrid, Spain
[†]Universidad Politécnica de Madrid, Spain
[‡]University Carlos III of Madrid, Spain
{mohamed.moulay, fernando.diez, vincenzo.mancuso}@imdea.org

***Abstract*—With the growth of the internet, current transport protocols are being re-evaluated to deal with traffic growth. QUIC is an evolving transport-layer protocol that has been developed to reduce Web latency, integrate security features, and enable a high-quality experience for mobile users. However, few systematic experimental studies have been carried out to evaluate users' service in a realistic mobile environment. In this work, we describe the design of experiments with QUIC and HTTP/3, which is based on QUIC, also considering benchmarks and variants in QUIC's implementations. We report our measurements collected from real mobile networks with the pan-European MONROE platform. Using data from the application, transport, and network layers in different wireless environments, we experimentally investigate and compare the performance of QUIC and HTTP/3 for static and mobile cellular users of several networks across Europe. Initial results show that QUIC-based operations are advantageous in case of users that move, and they are also strongly affected by the congestion control algorithm chosen for QUIC.**

***Index Terms*—QUIC; MBB; Experiments;.**

I. Introduction

HTTP prevails as the undisputed king of the Web with its two versions HTTP/1.1 and HTTP/2 adopted by the vast majority of websites. This is also clear in terms of Internet traffic [1] and in terms of the widespread support offered by all current browsers to HTTP/1.1 and HTTP/2, based on TCP and all its features. However, in recent years a new protocol has come into play with the name of QUIC (Quick UDP Internet Connections) as the foundation of upcoming HTTP3 [2]. QUIC is a new network transport protocol originally designed and proposed by Google, which operates on top of a simpler transport protocol, UDP, so as to bypass the limitations of legacy TCP/IP transport. QUIC itself is commonly seen as a transport-layer protocol, although it embeds encryption features and relies on conventional UDP transport.

QUIC eliminates significant TCP bottlenecks such as the need of initial TCP handshake mechanism, which takes one Round Time Trip (RTT) or two, in case of TCP data encryption via TLS. Indeed, Google has designed and implemented a novel encryption scheme for QUIC, similar to TLS, which couples connection establishment and key agreement within one RTT. Nevertheless, QUIC can start a connection in zero RTT, like UDP, by instantly sending encrypted application data to the server. This is possible when it previously has cached in a server certificate from a previous connection. Moreover, running on top of UDP, QUIC bypasses the head-of-line (HoL) issue of TCP in case of packet loss. For these reasons, in the long run, QUIC is expected to replace TCP and TLS in the Web [3].

Since offering video services and multimedia channels is the main trend in the Internet, which is stressful for the entire network infrastructure and in particular for Mobile broadband (MBB) networks, the evaluation of QUIC and HTTP/3 as an alternative to TCP and HTTP/1.1 or HTTP/2 is necessary in the steadily growing MBB environment [4]. This need motivates us to design, develop, and deploy experiments with QUIC and HTTP/3 on operational MBB networks.

Thus, we focus on QUIC and HTTP/3 performance figures in mobile environments, for which so far little experimental work exists. In fact, other works on assessing QUIC performance figures are currently growing, but only by studying basic properties in controlled environments. For instance, the authors of [5] use Google's server and the client to evaluate the performance of QUIC and TCP in wireless networks in a controlled environment although they have not tested the service offered by mobile operators and core networks. Similarly, the authors of [6] have QUIC and TCP in the kernel level through a gigabit LAN environment.

To assess QUIC and HTTP/3 in uncontrolled networks, we need, on the one side, an open and flexible implementation of QUIC, which allow for end-to-end testing and collecting statistics, possibly playing with QUIC's configuration. We use two open-source implementation of QUIC for this purpose: `flowsim`[1] and `Mvfst`[2]. We also resource to *qlog files*, to extract statistics [7]. On the other side, we also need objective information about MBB performance and reliability to provide an adequate QoE (Quality of Experience) to the end-user when using QUIC. Various initiatives have arisen recently to help in that direction, among which the US FCC's Measuring Broadband America initiative [4] and the European initiative MONROE [8], an open experiment-as-a-server platform meant to monitor and assess MBB performance. In our work, we leverage MONROE, to which we have direct access and which we have co-designed during the last years. Our results show that QUIC and TCP performance attributes over MBB are relatively similar, except QUIC reacts better to the vagaries of channel quality experienced by mobile nodes. However, with the application protocols, HTTP3 based on QUIC performs better than HTTP/1.1 and HTTP/2 based on TCP, although the choice of the QUIC's control algorithm sensibly impacts on the overall performance.

The main contributions of this paper can be summarized as follows: (i) we design Docker containers that run QUIC and TCP measurements as well as run HTTP variants, and integrate them in the MONROE platform; (ii) we explore

[1]https://github.com/paaguti/flowsim
[2]https://github.com/facebookincubator/mvfst

the performance of QUIC and HTTP/3 to download contents from the Web under heterogeneous conditions, in terms of geographical diversity and mobility; (iii) thanks to the flexibility of `flowsim`, which can act as a TCP or QUIC traffic generator as well as an HTTP server, we compare the performance of QUIC, TCP and compare HTTP variants; (iv) thanks to the flexibility of `Mvfst`, which has been developed by Facebook, we evaluate the impact of congestion control algorithms in QUIC; and (v) we generate a large open *qlog* dataset with our experiments with QUIC and HTTP/3 in real operational networks.

The rest of the paper is organized as follows. Section II presents an overview of related work. Section III offers background information on QUIC and the *qlog* file structure. Section IV discusses our measurement methodology and setup. Section V focuses on experimental results. Eventually, Section VI summarizes the findings of the paper and points out future research directions.

II. Related work

QUIC was first introduced as gQUIC (Google's version of QUIC), which led to an early investigation work [2], [9]–[13] assisting the performance of that specific implementation, mostly using Chromium as the open-source gQUIC server of choice in local environments. This means that the experimenters control various dimensions that can impact the overall performance, such as network conditions. Instead, we study QUIC in the wild of operational cellular networks in different countries. We can configure QUIC servers and client, but we cannot touch the network.

Existing studies, e.g., [3], [14], show a toss-up in terms of who is the better protocol QUIC or TCP. They dig deep into QUIC's features, such as 0-RTT connection establishment, congestion control, and removal of TCP's HOL block. Some of these studies conclude that QUIC outperforms TCP, while others show the opposite. These conflicts between results are due to the different tuning of application settings and machine kernel used in the experiments. Accordingly, recent works like [15] suggest to separate the QUIC protocol from the implementation since there exist multiple implementations produced by Google, Facebook, and Cloudflare. Moreover, many authors focus mostly on the gQUIC implementation of QUIC,[3] which is Google's implementation, and which differs from the proposed IETF version of QUIC,[4] as discussed by the authors of [15]. They sustain that most of the observed performance differences can be attributed to developer design and operator configuration choices when selecting the congestion control algorithm used in QUIC. They also point out that conducting new, experimental research from different implementation angles in production environments is needed. This is exactly what we present in this paper, leveraging two IETF-compliant open implementations of QUIC, namely `flowsim` and `Mvfst`, through which we are able to explore the impact of QUIC's configuration parameters in real cellular networks.

For our work, we leverage a large-scale measurement platform and focus on users connectivity through MBB networks only. Precisely, we use the MONROE platform, which has been designed with our contribution [16]. MONROE's main goal is to allow the collection of telemetries and the execution of experiments on operative mobile networks. This is made possible through the use of hardware nodes which are connected to different broadband operators and are locally running experiments uploaded by users and programmed by means of an automatic scheduler. MONROE is currently operating as an international alliance, formed after the consortium of a previous European project. MONROE provides multi-homed, autonomous, large-scale monitoring and evaluation of performance for mobile broadband networks in heterogeneous environments. Acquiring access to this platform allows for the deployment of vast measurement setups to collect data from operational MBB networks in various European countries. Differently from other approaches based on operator-driven quality-assessment campaigns [17], [18], or on traditional drive-by tests [19], MONROE offers an open platform for repeatable and traceable experiments. Besides, it offers open access to collected data, which refer to multiple operators, and includes device-level metadata, which is the key to use and possibly filter results without raising user's privacy concerns. This offers much richer data than what can be offered by crowdsourcing initiatives like, e.g., Netalyzer [20] and Haystack [21].

Thanks to MONROE, and unlike current approaches to QUIC measurements, we focus more on the performance assessment of QUIC under real cellular networks in different European countries, showing the impact of mobility on performance figures. We believe that showcasing how QUIC behaves under operational setting, and compared to TCP alongside different congestion control algorithms, offers a broader and more realistic image of QUIC than available studies, at least for what concerns the use of QUIC in mobile networks.

III. Background on QUIC operation and logging

A. The QUIC protocol in a nutshell

QUIC is a new evolving transport protocol originally proposed by Google to compete specific problems encountered while using TCP's HTTP/1.1 and HTTP2. QUIC provides similar guarantees to TCP in terms of reliable data delivery and strict flow control by combining TCP multiplexing and TLS into one transport protocol built on top of UDP [2].

To achieve that, QUIC employs features such as Multiplexed Streams [22], which eliminates the HOL problem found in TCP by extricating the ordering restrictions on packet delivery since in-order data delivery is done on the stream level rather than the connection level.

A key feature of QUIC is the shorter time to establish a secure connection within fewer RTTs than TCP/TLS. To achieve this, the TLS negotiation mechanism is part of the QUIC protocol by default, removing the need for a separate non-TLS TCP handshake to synchronize client and server [23].

These new features pave the way to implement HTTP/3 [24] as the upcoming web application protocol. HTTP/3 will evolve radically by avoiding the use of HTTP/2's stream metadata, by using a new header compression algorithm as well as by adopting a new prioritization scheme.

Another interesting feature of QUIC is that it can be rapidly deployed since it runs in the user space and not in the kernel space of the operating system. This allows researchers and developers to quickly test and deploy and evaluate flow-control and congestion-control alternatives.

[3] https://www.chromium.org/quic

[4] https://quicwg.org

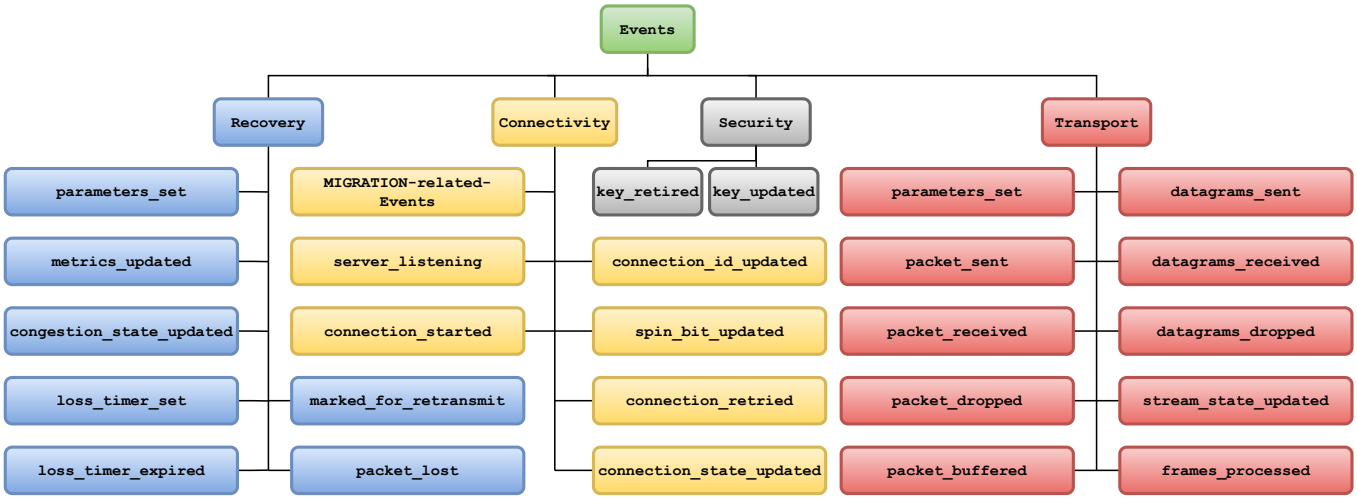


Fig. 1: QUIC events used in *qlog*

As for the loss and recovery mechanism, which is run on top of connection-less UDP features, QUIC introduces monotonically-increasing packet numbers to differentiate between new and retransmitted data, which significantly simplifies RTT measurements [23].

However, QUIC's diversification of features is a double-edged sword that might contribute to generate performance inconsistencies across implementations. This raises the need to investigate QUIC's performance under different real cellular networking environments, with different congestion control algorithms.

B. QUIC logging

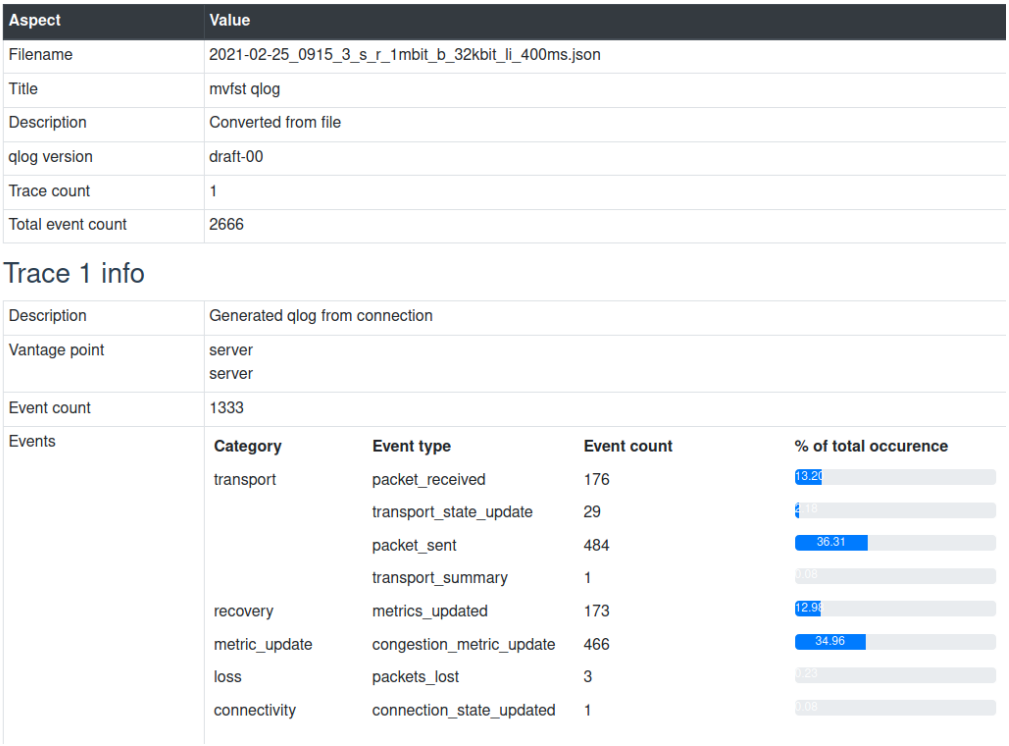
QUIC is characterized by a series of messages which govern the end-to-end interaction between a server and a client. As such, QUIC counts with a predefined set of events which can be captured and analyzed offline. Well-established protocols like TCP rely on specific analysis and debugging tools that directly present the protocol information as it would be seen externally by another entity. Existing tools for TCP rely on pcap files, which simply capture IP packets (or even lower layer frames) and read into protocol headers to reconstruct TCP flows and events. This approach, however, prevents the access to state variables that are internal to the communication and are not conveyed through protocol headers, although they can help the inspection of more intricate elements, such as congestion control algorithms. Thus, differently from the case of TCP, debugging a protocol as complex as QUIC, which is not a legacy transport protocol and comes with native encryption, is not feasible through the collection of legacy pcap files [7].

To this aim, the QUIClog project [7] was created to define a specific format, *qlog*, to organize and record, in so-called *qlog* files, different information captured during an exchange over QUIC. Logging occurs within the QUIC implementation, at server side. However, it must be said that several implementations of HTTP/3, which runs over QUIC at an application level, have managed to capture logs on the client side as well. As our study will center on the evaluation of QUIC's transport-based capabilities, this possible scenario will not be explored in the paper. Although QUIC developers are free to introduce additional information in *qlog* [25], conferring to the log a certain degree of extensibility, some *qlog* fields are reserved and predefined, and a general basic indexing structure must be guaranteed to parse information correctly.

A *qlog* file describes protocol events, not just QUIC events, according to the scheme shown in Fig. 1. Each event record must include, as specified in [26], a timestamp indicating the time at which the event was registered, a field describing the type of event associated to the record, and finally, the data registered. Events can be grouped in categories, which although not mandatory are recommended to favour high-level filtering of logs. It is also essential to note that a *qlog* can trace activity from different protocol stacks, thanks to the fact that a field is reserved to indicate the type of protocol the event is related to, making it possible to capture both TCP and QUIC information.

Regarding event definitions related to QUIC traces, using *qlog* allows to record information related to the connection and key-sharing processes, as well as the data-exchange process. It does so not only gathering information about the frames sent and received from the server-side, but also by including transport metrics relevant for the communication, such as RTT and congestion window updates. As a matter of fact, these values, specially time delays, are aggregated and classified under the recovery category, whose event types are specified in a generic way to ensure that diverse congestion and recovery schemes can be covered by the logging format. It is mainly thanks to the information gathered in the recovery category that it is possible not only to evaluate QUIC's performance over time but also to fine-tune the protocol, as congestion and loss detection parametrization is also recorded in *qlog* files. To improve the interpretability of results, a visualization tool, `qvis`,[5] is also available for users. As pictured in Fig. 2, `qvis` allows for the import of logging files and the graphical display not only of the packet exchange between nodes but also of aggregated statistics that can be extracted from *qlog* events, such as packet loss percentage, as well as event count and percentage of occurrence. For all the aforementioned, *qlog* files are currently the main source of insight to understand the behaviour of QUIC.

[5]https://qvis.quictools.info



Aspect	Value
Filename	2021-02-25_0915_3_s_r_1mbit_b_32kbit_li_400ms.json
Title	mvfst qlog
Description	Converted from file
qlog version	draft-00
Trace count	1
Total event count	2666

Trace 1 info

Description	Generated qlog from connection
Vantage point	server server
Event count	1333

Events

Category	Event type	Event count	% of total occurence
transport	packet_received	176	13.20
	transport_state_update	29	2.18
	packet_sent	484	36.31
	transport_summary	1	0.08
recovery	metrics_updated	173	12.98
metric_update	congestion_metric_update	466	34.96
loss	packets_lost	3	0.23
connectivity	connection_state_updated	1	0.08

Fig. 2: Example of `qvis` statistics view

IV. Experimental Methodology

The QUIC protocol has been evaluated mainly in controlled and simulated scenarios, which is useful to understand the basic behavior of the protocol. Here, we want to capture the complexity of operational cellular networks, where radio conditions impact performance and can vary immensely in a brief period. As such, we want to shed light on the behavior of QUIC in the context of realistic mobile scenarios. Moreover, aware of the difficulties for researchers to count with a testing environment that allows them to experiment over operational networks and use existing infrastructure, we wanted to contribute by building and releasing a *qlog* dataset with the files extracted from our tests.[6]

In addition, we take into consideration that multiple open-source versions of the QUIC protocol are available, each of them offering distinctive tuning possibilities for the configuration of QUIC instances. This opens the possibility of tackling the comparison between the most well-known and widely adopted implementations, currently those being `flowsim` and `Mvfst`. It also opens the possibility to test the congestion control algorithms currently under evaluation at IETF for QUIC, which also entails the need to compare QUIC to TCP.

A. Methodology

We study the case of downloads from a network server to a mobile node and the case of webpage retrieval with HTTP variants. The methodology chosen to assess experimentally QUIC's configurations follows an Outside-In approach, where we first center on external information that can be extracted from the experiment, such as its duration or the throughput of the download. To do so, we frame the experiments in the context of cellular networks, and develop Docker containers compatible with the MONROE platform. Experiments are defined in the form of Docker containers embedding `flowsim` and `Mvfst` clients. They run on MONROE nodes, which are network probes, i.e., devices accessing the Internet through three operational commercial MBB networks, using three distinct cellular interfaces. The use of containers favours an efficient use of the MONROE node's resources and ensures that the experiment execution does not compromise the configuration of the node or future experiment runs.

We test and certify our containers with the MONROE authority, and obtain access to the MONROE automatic experiment scheduler, through which we deploy recurrent experiments at client side. Specifically, the scheduler deploys our containers with our transport clients on selected MONROE nodes when the time to run the experiment comes.

The server, which acts as a traffic generator and an event logger, runs in a lab machine in which we implement `flowsim` and `Mvfst` and tune the structure of *qlog* files to collect the desired flow statistics of both QUIC and TCP downloads. Fig. 3 shows our experimental scenario, which includes the tools needs for storage and retrieval of measurements.

Since QUIC allows multiple parallel flows, and the fact that the selected tools can be run in multiple instance on the servers, and hence allow to receive multiple incoming requests in parallel with different server configurations, entail the consideration of two different types of download experiments: one in which the client issues download requests sequentially, and thus does not have to share network resources between flows, and a second one, in which multiple download requests are triggered in parallel from the client against different ports in the server (which corresponds to one client connected to multiple servers on the same machine). With this, we can test multiple configurations on QUIC in parallel, and also test their fairness. In addition, `flowsim` offers the possibility to mimic HTTP (versions 1.1, 2 and 3), so that we use it to reproduce the behavior of webpage

[6]Dataset available at https://doi.org/10.5281/zenodo.4602217

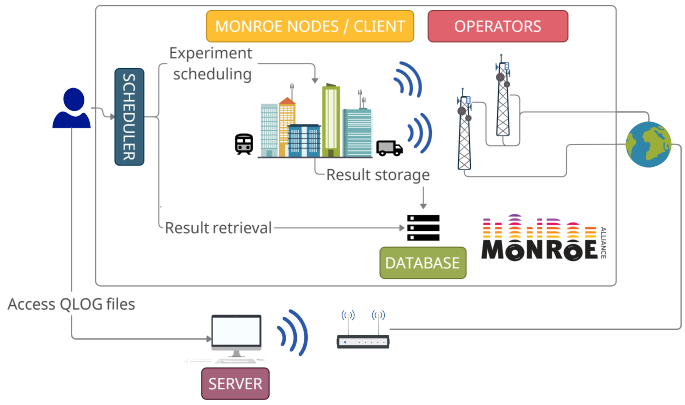


Fig. 3: Experimental scenario

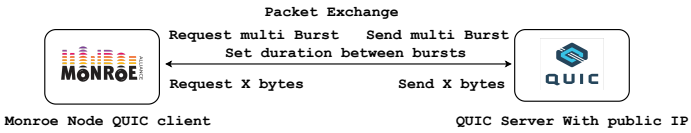


Fig. 4: `flowsim` MONROE platform setup

retrieval with TCP and QUIC.

We repeat all types of experiments sequentially from multiple MONROE nodes and from each of the cellular interfaces available on the nodes (note that each MONROE node has three interfaces connected to three different MBB operators).

Although download/page retrieval time and throughput (or TCP goodput) measurements can provide an overview of the experiment's performance, it does not explain the reasons behind the results gathered, due to the complexity of the QUIC and TCP protocols. We then first analyze the logs of our Docker containers for aggregate download statistics, and then move to the analysis of *qlog* files redacted by our experimental server, which allows to evaluate the behavior of the transport protocol during the download.

Indeed, the inspection of experiment results and *qlog* files allows us to study the evolution of transport-related metrics. From the user point of observation, in the Docker container, we collect flow statistics such as the (webpage) download time, which is equivalent to estimate the average throughput during the download. Instead, with the *qlog* files generated at the server, we study the behavior over time of experienced RTT and throughput, and how the congestion window evolves and drives the volume and speed of data downloaded.

B. Setup

Our Docker containers come with a QUIC/TCP client request file download from a server located in our lab that runs a QUIC/TCP traffic generator, as shown in Fig. 3.[7]

A client in the container interacts with our server as exemplified in Fig. 3 for the case of QUIC with `flowsim`. We have opened a limited number of ports on a few machines in our lab and across Europe in labs of the partners of the MONROE Alliance. All servers have public IP address, so that a client on a MONROE node, and hence passing through a cellular network and the public Internet, can request multiple bursts of traffic (i.e., file downloads).

[7]Our Docker containers are publicly available for interested experimenters at https://hub.docker.com/repository/docker/07777/fullquic and https://hub.docker.com/r/7466291/zmq-mvfst-tperf

We use clients and servers in Spain, Italy, Norway and Sweden. For the clients, we distinguish between static nodes and mobile nodes, the latter being MONROE nodes installed on trains in Norway and busses in Sweden. Static nodes have high-quality connectivity, while mobile nodes can suffer low quality and even outages. In all cases, the server-side of our experiment is connected to a gigabit LAN directly connected, in turn, to a multi-gigabit metropolitan network.

The Docker container at the client requests 100 kB to 1 MB per download burst in the experiments with `flowsim`, to test Web downloads, and set a time interval of 15 seconds for experiments with `Mvfst`, to emulate streaming cases. we repeat the download 10 times in each experiment from each node, alternating downloads to pause times during which we select a different cellular interface. The pause time is set to 10 seconds, and a data burst is interrupted after 60 seconds in case the file download or the page retrieval is not completed before. This is needed in order to avoid hanging connections during the experiments. We collect the download time in seconds for all requests as well as the underling *qlog* events.

In the experiments with `Mvfst`, we configure the download of each file several times, using a different congestion control algorithm every time, and alternating between single downloads and multiple parallel downloads. The Docker container with `Mvfst` uses `tperf` to tell the QUIC server which congestion control protocol has to be used. Indeed, the fact that QUIC runs in user space, makes it possible to reconfigure the server on the fly. We experiment with Cubic, BBR, COPA and Newreno congestion control algorithms, which are encoded in `Mvfst`. Instead, `flowsim` only implements Cubic.

For what concerns COPA, a fairly new congestion control scheme that follows the delay-based approach undertaken by BBR, we remark that we have not played with its optimization, and we have used a default COPA's latency factor of 0.5, which corresponds to an equilibrium between queueing delay and throughput [27].

V. Results

We start first by comparing pure TCP with pure QUIC, followed by comparing application protocols HTTP (i.e., HTTP/1.1), HTTPS (i.e., HTTP/2), and HTTP3 (i.e., `flowsim`'s implementation of HTTP3 with QUIC).

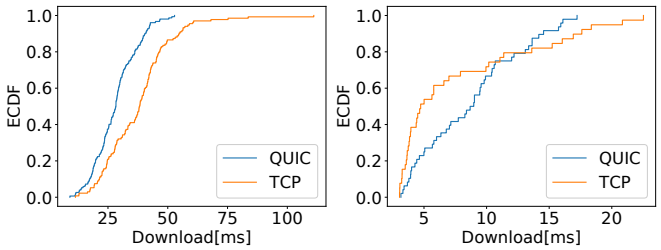
We only report a small yet characteristic subset of the results we have collected and start first by showcasing the results gathered with `flowsim` and then move to `Mvfst`. For the interested experimenters, all the results are accessible online.[8]

A. Assessment of QUIC and HTTP/3 performance

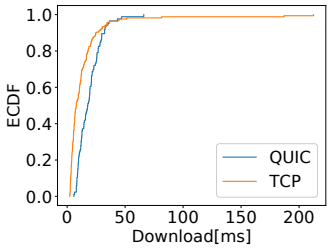
Here we report the results of our `flowsim` experiments for the evaluation of QUIC and HTTP/3 vs TCP and older HTTP versions, using a single stream per download.

Fig. 5 and Fig. 6 show that empirical distribution of the download time for traffic burst of 1 MB. The considered experiments were run from static nodes in Spain, Norway and Sweden. Fig. 5 contains separated statistics for the three countries. Differences across countries are normal also because we put together the results of three operators per country, each with different latency to the servers. However,

[8]https://github.com/Mohmoulay/QUIC-MedNetCom

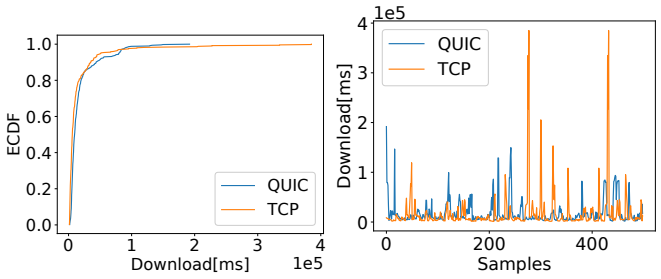


(a) Spain (b) Norway



(c) Sweden

Fig. 5: Download time comparison between QUIC and TCP, with `flowsim` clients in Spain, Sweden and Norway (download size of 1 MB)



(a) ECDF (b) Samples

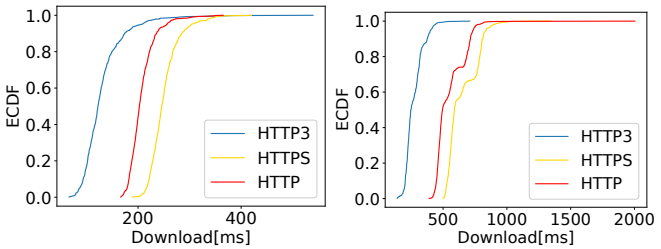
Fig. 6: Download time comparison of QUIC and TCP with `flowsim` on a public bus in Sweden (1 MB downloads)

for each country we use a local server in that same country. The figure also report TCP statistics for the same kind of experiment. It is interesting to see that QUIC performs better than TCP in Spain, but not in Norway and Sweden. The differences are however not huge.

Fig. 6 reports download time statistics, and the collected samples, for the case of mobile clients located on busses in Sweden. The distribution of results shows practically no differences between QUIC and TCP. However, TCP tends to suffer occasionally and reports higher extreme values. This means that QUIC reacts better to channel outages than TCP.

Next, we turn our attention to the application protocols, using the HTTP emulation mode of `flowsim`, and we change the download size to 100 KB to match the size of real webpages. We can notice a difference in download time, as observed in Fig. 7 for Spain (with static clients) and Sweden (with clients on the move). In this case, HTTP3, which is based on QUIC, offers a clear advantage, especially with respect to HTTP/2, due to the reduced connection establishment time.

We also collect *qlog* files for plain QUIC and HTTP3 experiments. Here we show the case of the mobile Swedish nodes. In Fig. 8, we report RTT, throughput, congestion



(a) Spain (b) Sweden

Fig. 7: Page download time comparison between HTTP versions with `flowsim` clients in Spain (static) and Sweden (mobile), with page size of 100 kB

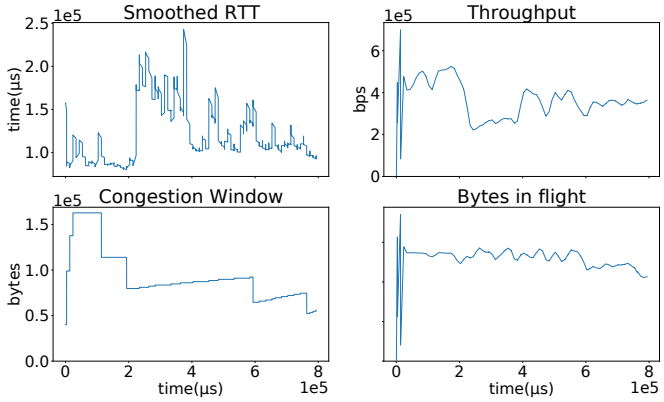


Fig. 8: Time series for for a QUIC download from a mobile `flowsim` client in Sweden (*qlog* events, 100 kB downloads)

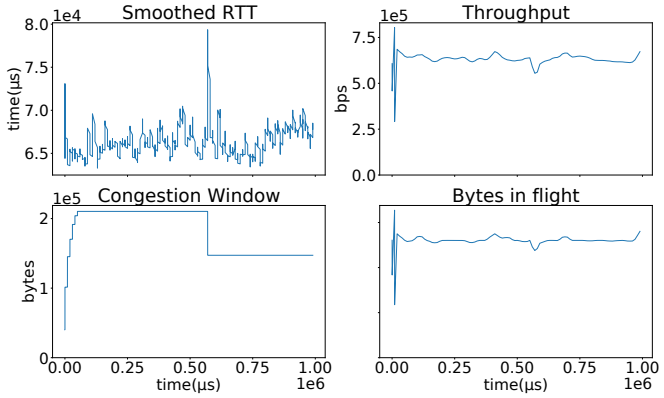


Fig. 9: Time series for a webpage download from a mobile `flowsim` client in Sweden (*qlog* events, 100 kB downloads)

window and number of bytes in flight as reported in the *qlog* files for a specific experiment lasting about one second. From the time series Fig. 8, we can see how the `flowsim` implementation of QUIC adapts fast to delay variations (due to the mobility of the client), and the resulting volume of bytes in flight is maintained practically constant after a short transient phase.

Similarly, for the case of HTTP/3 with a mobile client, Fig. 9 shows that a QUIC-based Web browsing is quite robust to fast channel quality variations.

B. Assessment of Congestion Control Variants in QUIC

Here we evaluate the impact of congestion control on QUIC and how it behaves over time in the presence of

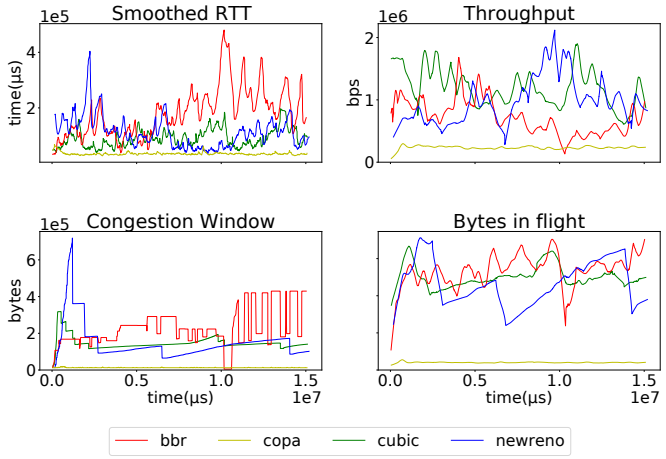


Fig. 10: Sequential time series in Spain with three download streams (*qlog* files of `Mvfst` experiments)

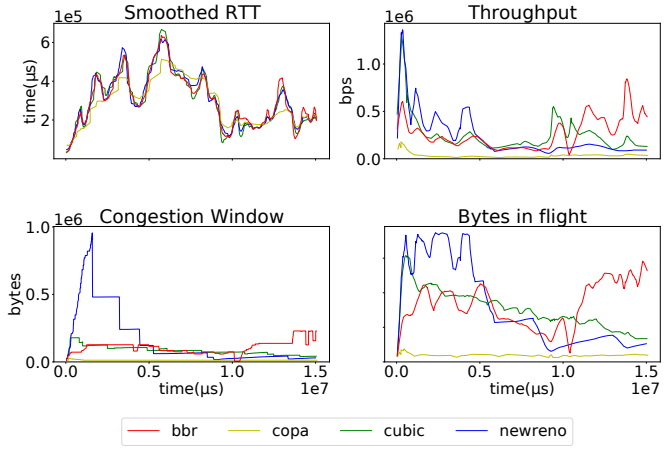


Fig. 11: Parallel time series in Spain with three download streams (*qlog* files of `Mvfst` experiments)

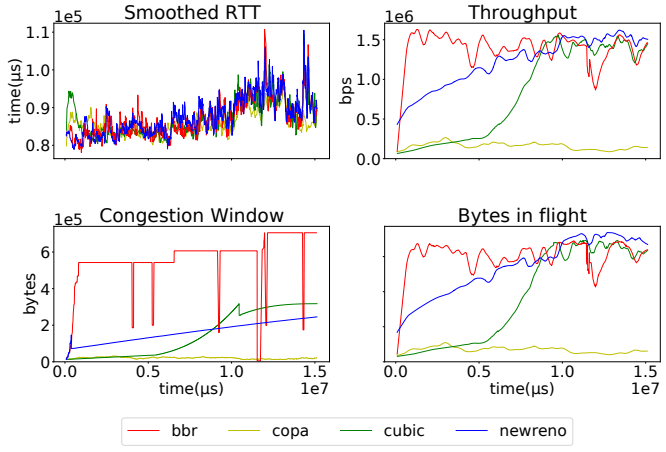


Fig. 12: Parallel time series in Norway (with a static `Mvfst` client) with three download streams

multiple downloads streams. We use `Mvfst` for the set of experiments described in this subsection, because it allows to test several congestion control schemes. Results with `Mvfst` running Cubic are very similar to the ones obtained with `flowsim`, so we do not include `flowsim` experiments here. Instead, we compare sequential vs parallel experiments: in the sequential ones, the congestion control scheme is switched from one set of experiments to the next, while in parallel experiments, the various congestion control schemes are run in parallel on different instances of the QUIC servers, using different ports. This also allows us to evaluate the fairness among QUIC variants.

Figs. 10 and 11 report the results for a download stream, performed sequentially or in parallel, respectively, from the same static MONROE client in Spain. The figures include statistics extracted from the *qlog* files, and refer to the case of three streams per download (mimicking the download of multiple media channels), although results with a different number of streams are similar. The figures also report results for four congestion control schemes, the first (Cubic) being the one also used in the experiments with `flowsim`.

In the *sequential* case of Fig. 10, we can see that the RTT observed are different for the various congestion control cases, because they are run one after the other. Instead, in the *parallel* case of Fig. 11, the RTT curves are very similar, because the flows with different congestion control are run in parallel. The curves are not exactly the same because each IP packet is routed independently, which results in small fluctuations of latency from one packet to the other.

In Fig. 10, we can observe that Cubic is the congestion control scheme that yields the highest throughput, on average, while BBR adapts better and faster to channel variations. Newreno achieves the highest pick rates, but also suffers the higher variations of throughput and in the use of buffer space, witnessed by the least stable values in the number of bytes in flight. COPA is by far the least performing scheme, as visible, e.g., in the low values reached by its congestion window.

In Fig. 11, with parallel tests, we can observe that BBR is more stable and adapts quicker to network conditions than the other schemes. This is in contrast with the operation of classic schemes like Cubic an Newreno, which hardly come close to the maximum congestion window size after a change in network conditions. This eventually results also in a better average throughput for BBR, although Cubic obtains comparable results.

With better and more stable network conditions, like shown in Fig. 12 for a static parallel experiment with a client in Norway, BBR clearly shows its ability to jump fast to better operational conditions with respect to the other schemes. However, it is also interesting to note that Cubic, BBR and Newreno are essentially fair with each other, while COPA exhibits strong limitations.

The fact that BBR is the scheme that succeeds the most at maintaining a sustained throughput value across executions is also visible from experiments with mobile nodes. For instance, Figs. 13 and 14 show that BBR can introduce oscillations in the congestion window which eventually result in stable and high throughput and number of bytes in flight. This is true both in case of sequential and parallel experiments, which means that the result is not a byproduct of the coexistence of multiple download flows.

Regarding COPA, we remark that we have not optimized its parameters, because that was out of the scope of our measurement study. We leave for future work the investigation on the impact of COPA's configuration on QUIC performance.

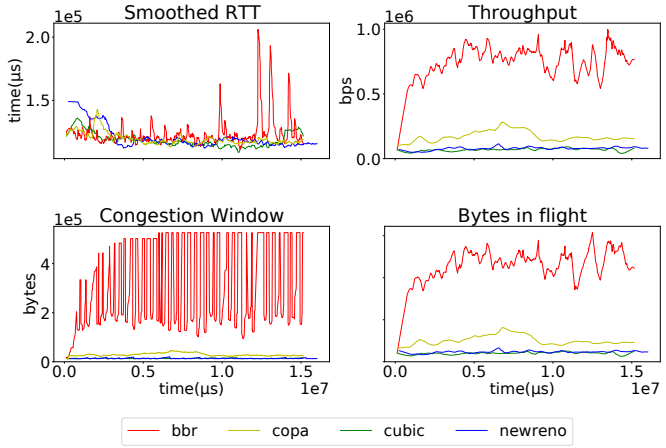


Fig. 13: Sequential time series in Sweden (with a mobile `Mvfst` client) with three download streams

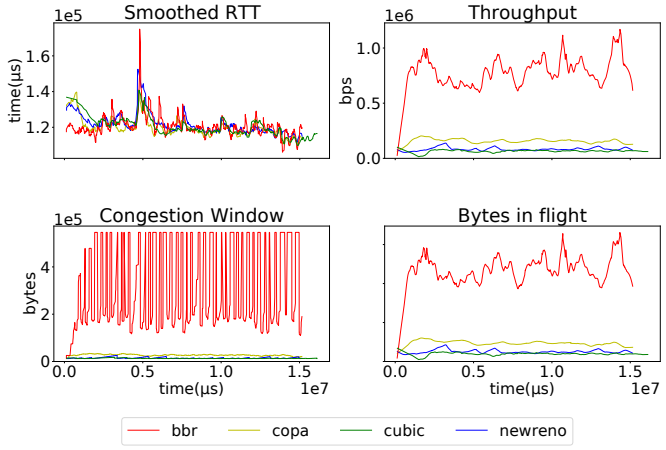


Fig. 14: Parallel time series in Sweden (with a mobile `Mvfst` client) with three download streams

VI. Conclusion

In this work, we have evaluated the performance of QUIC and HTTP/3 under different cellular network conditions and with multiple congestion control algorithms. We have leveraged the MONROE platform and two open QUIC implementations, namely `flowsim` and `Mvfst`, and employed Docker containers and *qlog*. Containers and raw measurements have been made available online.

By analyzing container logs and *qlog* files, we have observed that QUIC is advantageous over TCP for what concerns HTTP applications. In addition, the congestion control algorithm, especially under mobility, strongly impacts the QUIC's overall performance, and BBR yields the more stable performance figures to mobile users.

Our tools and methodology can be used in the future for a broader and continuous assessment of ever-evolving QUIC-based protocols.

Acknowledgements

This work is partially supported by the Region of Madrid, Spain, through the TAPIR-CM project (S2018/TCS-4496).

References

[1] K. Nepomuceno, I. N. d. Oliveira, R. R. Aschoff, D. Bezerra, M. S. Ito, W. Melo, D. Sadok, and G. Szabó, "QUIC and TCP: A performance evaluation," in *IEEE ISCC*, June 2018.

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC transport protocol: Design and internet-scale deployment," in *Proceedings ACM SIGCOMM*, 2017.

[3] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same standards, different decisions: A study of QUIC and HTTP/3 implementation diversity," in *Proceedings of ACM EPIQ*, 2020.

[4] FCC, "2013 Measuring Broadband America February Report," FCC's Office of Engineering and Technology and Consumer and Governmental Affairs Bureau, Tech. Rep., 2013.

[5] P. K. Kharat, A. Rege, A. Goel, and M. Kulkarni, "QUIC protocol performance in wireless networks," in *Proceedinfs of ICCSP*, 2018.

[6] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the QUIC protocol in Linux kernel," in *Proceedings of ACM MSWIM*, 2018.

[7] R. Marx, W. Lamotte, J. Reynders, K. Pittevils, and P. Quax, "Towards QUIC debuggability," in *Proceedings of ACM EPIQ*, 2018.

[8] O. Alay, A. Lutu, M. Peón-Quirós, V. Mancuso, T. Hirsch, K. Evensen, A. Hansen, S. Alfredsson, J. Karlsson, A. Brunström, A. Safari Khatouni, M. Mellia, and M. Ajmone Marsan, "Experience: An open platform for experimentation with commercial mobile broadband networks," in *Proceedings of ACM Mobicom*, 2017.

[9] P. Biswal and O. Gnawali, "Does QUIC make the Web faster?" in *Proceedings of IEEE GLOBECOM*, 2016.

[10] G. Carlucci, L. De Cicco, and S. Mascolo, "HTTP over UDP: An experimental investigation of QUIC," in *Proceedings of ACM SAC*, 2015.

[11] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at QUIC: An approach for rigorous evaluation of rapidly evolving transport protocols," in *Proceedings of ACM IMC*, 2017.

[12] M. Nguyen, H. Amirpour, C. Timmerer, and H. Hellwagner, "Scalable high efficiency video coding based HTTP adaptive streaming over QUIC," in *Proceedings of ACM EPIQ*, 2020.

[13] J. Rüth, K. Wolsing, K. Wehrle, and O. Hohlfeld, "Perceiving QUIC: Do users notice or even care?" in *Proceedings of ACM CoNEXT*, 2019.

[14] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?" in *Proceedings of IEEE ICC*, 2017.

[15] T. A. B. Alexander Yu, "Dissecting performance of production QUIC," in *Proceedings of ACM EPIQ*, 2021.

[16] V. Mancuso, M. Peón Quirós, C. Midoglu, M. Moulay, V. Comite, A. Lutu, Özgü Alay, S. Alfredsson, M. Rajiullah, A. Brunström, M. Mellia, A. Safari Khatouni, and T. Hirsch, "Results from running an experiment as a service platform for mobile broadband networks in europe," *Computer Communications*, vol. 133, pp. 89 – 101, 2019.

[17] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, S. Venkataraman, and J. Wang, "A first look at cellular network performance during crowded events," in *Proceedings of SIGMETRICS*, 2013.

[18] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z.-M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of LTE: Effect of network protocol and application behavior on performance," in *Proceedings of ACM SIGCOMM*, 2013.

[19] Tektronix, "Reduce drive test costs and increase effectiveness of 3G network optimization," Tektronix Comm., Tech. Rep., 2009.

[20] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the edge network," in *Proceedings of ACM IMC*, 2010.

[21] N. Vallina-Rodriguez, "Illuminating the third party mobile ecosystem with the Lumen privacy monitor," in *Proceedings of FTC PrivacyCon*, 2017.

[22] P. Qian, N. Wang, and R. Tafazolli, "Achieving robust mobile Web content delivery performance based on multiple coordinated QUIC connections," *IEEE Access*, vol. 6, pp. 11 313–11 328, 2018.

[23] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is QUIC? Provable security and performance analyses," in *IEEE Symposium on Security and Privacy*, 2015.

[24] M. Bishop, "Hypertext transfer protocol version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-34, Feb. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34

[25] R. Marx, "QUIC and HTTP/3 event definitions for qlog," Internet Engineering Task Force, Internet-Draft draft-marx-qlog-event-definitions-quic-h3-02, Nov. 2020, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-marx-qlog-event-definitions-quic-h3-02

[26] ——, "Main logging schema for qlog," Internet Engineering Task Force, Internet-Draft draft-marx-qlog-main-schema-02, Nov. 2020, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-marx-qlog-main-schema-02

[27] V. Arun and H. Balakrishnan, "COPA: Practical delay-based congestion control for the Internet," in *Proceedings of ANRW*, 2018.