This is a post-peer-review, pre-copyedit version of an article published in *Journal of network and systems management*. The final authenticated version is available online at:
http://dx.doi.org/10.1007/s10922-020-09563-8

On the Performance of QUIC over Wireless Mesh Networks

Jawad Manzoor[1] [2], Llorenç Cerdà-Alabern[2], Ramin Sadre[3], Idilio Drago[4]
Air University[1]
Universitat Politècnica de Catalunya[2]
Université catholique de Louvain[3]
Universita degli Studi di Torino[4]
*Corresponding author: Jawad Manzoor, jawad.manzoor@mail.au.edu.pk

July 2020

Abstract

The exponential growth in adoption of mobile phones and the widespread availability of wireless networks has caused a paradigm shift in the way we access the Internet. It has not only eased access to the Internet, but also increased users' appetite for responsive services. New protocols to speed up Internet applications have naturally emerged. The QUIC transport protocol is one prominent case. Initially developed by Google as an experiment, the protocol has already made phenomenal strides, thanks to its support in Google's servers and Chrome browser. Since QUIC is still a relatively new protocol, there is a lack of sufficient understanding about its behavior in real network scenarios, particularly in the case of wireless networks. In this paper we present a comprehensive study on the performance of QUIC in Wireless Mesh Networks (WMN). We perform a large scale measurement campaign on a production WMN to compare the performance of QUIC against TCP when retrieving files from the Internet. Our results show that while QUIC outperforms TCP in wired networks, it exhibits significantly lower performance than TCP in the WMN. We investigate the reasons for this behavior and identify the root causes of the performance issues. We find that some design choices of QUIC may penalize the protocol in WiFi, e.g., uncovering sub-optimal interactions of QUIC with MAC layer features, such as frame aggregation. Finally, we implement and evaluate our solution and demonstrate up to 28% increase in throughput of QUIC.

1 Introduction

The Internet is evolving from the perspective of both usage and connectivity. Wireless networks (in particular WiFi) are nowadays a commonplace across the board including homes, offices, shopping malls, restaurants, hospitals and university campuses. Wireless Mesh Networks (WMN) [1] are getting popular, and some examples include Guifi.net, MadMesh, Merkai and Google WiFi. Wireless traffic continues to grow at an unprecedented rate and it is predicted in a recent Cisco white paper [2] that wireless and mobile traffic will exceed that of PCs, and will comprise more than 63 percent of the total IP traffic by 2021.

While wireless networks provide mobility and ease of access to the Internet, they are also prone to errors due to obstacles, moving objects, weather conditions, noise and interference with other wireless sources etc. Nonetheless, technological advancements have allowed to tackle these issues and provide good quality of experience. On the one hand, WiFi technology (based on the IEEE 802.11 standards) has undergone a lot of improvements [3]. For instance, the IEEE 802.11n standard, predominant by now, allows coding schemes (MCS) that support data rates up to 600 Mbps. On the other hand, there have been many efforts to improve upper layer protocols so to better deal with issues that are inherent of the wireless links.

Such improvements have naturally involved the transport layer. The Transmission Control Protocol (TCP), being the most widely used transport protocol in the Internet, has undergone many improvements over the years. These improvements have helped TCP to efficiently exploit advanced WiFi transmission methods too. Yet, TCP has some inherent issues, which limit its

performance regardless of the used medium. Thus, new transport protocols for the Internet have been proposed, and among them we have witnessed the successful deployment of QUIC.

QUIC started as an experimental protocol designed by Google. It is a transport protocol that aims to provide a better user experience by reducing latency at the application layer. In a recent measurement study [4], it was estimated that around 7% of the Internet traffic is carried by QUIC. Two of the most popular Internet services, Google Search and YouTube, run over QUIC. Other companies, such as Akamai, have also started to provide support for QUIC. In 2016 the IETF chartered the QUIC Working Group, which has used Google's QUIC as starting point for standardizing the protocol, an effort that is still ongoing by the time of writing.

Since QUIC is in its early stages and is gradually evolving, there are still many open questions about its deployment in real network scenarios. Prior studies [5, 6, 7, 8] on the performance of QUIC mostly focus on wired networks, simple lab-based WiFi test-beds or simulations. In contrast, we here present a comprehensive study of the performance of QUIC by performing a measurement campaign on a production WMN in Barcelona, Spain. We rely on active probes implemented in Raspberry Pi devices that are deployed at different nodes of the WMN and compare the performance of QUIC against TCP. The results show that QUIC exhibits significantly lower performance than TCP in the WMN. In our prior work we identified how packet pacing in QUIC affects the frame aggregation at MAC layer [9]. In this paper we extend our prior work and perform in-depth analysis of congestion control and loss detection algorithms of QUIC protocol to find out the root causes of its performance degradation in WiFi networks. We reveal the factors that impact the performance of QUIC in WiFi and implement possible solutions as a proof-of-concept, showing that they significantly improve QUIC performance. We notice that the throughput of the raspberries presented in this paper are significantly higher than those shown in our previous work [9]. This is because some important links of the mesh were improved before the data gathering campaign used in this paper.

In sum, the main contributions of our work are as follows:

- We provide the first measurement study of QUIC in a production WiFi community mesh network, comprising heterogeneous WiFi devices in an urban area, having a high diversity of quality in the links, and a large variety of real-world network conditions;
- We explore how advanced 802.11 standard features (e.g., frame aggregation) are impacted by different configurations of QUIC parameters;
- We discover that high delay variations, which are common in wireless networks, negatively impact the congestion control and loss detection algorithms of QUIC;
- We implement several optimizations as a proof-of-concept, showing up to 28% increase in QUIC throughput in the WMN.

This paper is organized as follows. Section 2 describes the background of QUIC protocol and also presents the work related to transport and MAC layer enhancements for WiFi. In Section 3 we give details about the methodology and describe the measurement environment we used to test the performance of the QUIC. Section 4 presents the results of our measurements. In Section 5 we provide an in-depth analysis of the results and in Section 6 we present the optimizations for QUIC and their evaluation. Finally, in Section 7 we conclude our work.

2 Background and Related Work

2.1 QUIC History

QUIC started as an experimental protocol by Google in 2012 to solve some of the inherent issues of TCP. The design goals of QUIC include rapid evolution and deployment, multi-streaming, latency reduction, flexible congestion control, connection migration and resilience to NAT-rebinding [4]. To meet these goals, QUIC is developed as a user-space transport protocol running on top of UDP. It provides several cross-layer enhancements, covering the weaknesses of TCP for transporting web content.

In 2016 the IETF chartered the QUIC working group which used Google's QUIC as starting point. Several important changes have been introduced during the last 2 years. TLS 1.3 has replaced Google's crypto libraries. A new header compression scheme (QPACK) has been designed to replace HPACK, the HTTP/2's header compression scheme. Several implementations of IETF and Google QUIC are currently available and a comprehensive list can be found at [10]. In this paper we use Google QUIC v43 which is the latest available version at the time of our research and is supported in the Chromium browser.

2.2 QUIC congestion controller

Describing all design decisions in QUIC is out of the scope of this paper. Some basic features of QUIC are explained in Appendix A. We however list key aspects of the protocol in the following, which will help to understand our measurements.

QUIC has a pluggable congestion controller which allows different algorithms to be used. QUIC currently provides implementations for Cubic, NewReno and BBR. Here again, we refrain from explaining all details of the various congestion control algorithms, listing only key characteristics that are needed to understand results that follow. All information that is provided next is obtained by analyzing the source code of QUIC v43 implemented by Google.

- Slow start algorithm
  Like with the TCP congestion control algorithms, QUIC slow start (or the *exponential growth* phase) is used to quickly converge to the available bandwidth of the end-to-end path between the client and the server. The implementation of QUIC tested in this paper exits the slow start phase when either a packet loss or an increasing delay is detected. To detect increases in the delay, which is assumed to be an indication of queuing in the network, QUIC compares the minimum RTT of the current burst of packets to the minimum RTT of the connection. If the delay is larger than a certain threshold, the slow start phase ends. The tested QUIC implementation uses a *slow start delay factor* of `1/8`, which implies that the slow start will exit if the RTT has increased by a factor of `1.25` when compared to the minimum RTT.

- Acknowledgment modes
  Google's implementation of QUIC includes two acknowledgment modes:
  – **TCP_ACKING**: This mode is similar to TCP delayed acknowledgment, in which an ACK is generated for every 2 received packets in accordance with RFC 1122. This is the default mode of QUIC in Chromium at the time of writing;
  – **ACK_DECIMATION**: In this mode acknowledgments are delayed up to a maximum of 10 packets and a cumulative ACK is generated. The maximum duration for which the ACK can be delayed is 25 ms and the actual `delay_time` is calculated on the fly as the minimum between `max_delay_time` and 1/4 of minimum RTT observed during the connection. The `ACK_DECIMATION` is only considered after at least 100 packets have been received to avoid interfering with slow start. This mode is disabled by default.

- Loss detection algorithm
  As said above, QUIC sends an acknowledgement every two segments. It detects losses by tracking the packets that are in-flight and unacknowledged. When it receives an ACK for packets sent later than unacknowledged packets, it starts a timer (based on the sending time of the possibly lost packet) and declares the packet as lost if its acknowledgement is not received within a particular period. QUIC uses a *time reordering factor* of `1/8` (fraction of the RTT). That is, the packet is considered lost if the timer reaches `1.25 * RTT` from the packet sending time.

- Packet pacing
  QUIC v43 includes a packet pacing mechanism that aims to reduce sending bursts of packets by introducing delay between consecutive packets. Reducing traffic burstiness is known to prevent congestion and, as a consequence, reduce the undesirable effects of packet loss. The pacing rate is calculated using the cwnd and the smoothed RTT. During slow start, the pacing rate is $2 * cwnd/smoothed\ RTT$ and during congestion avoidance it is $1.25 * cwnd/smoothed\ RTT$.

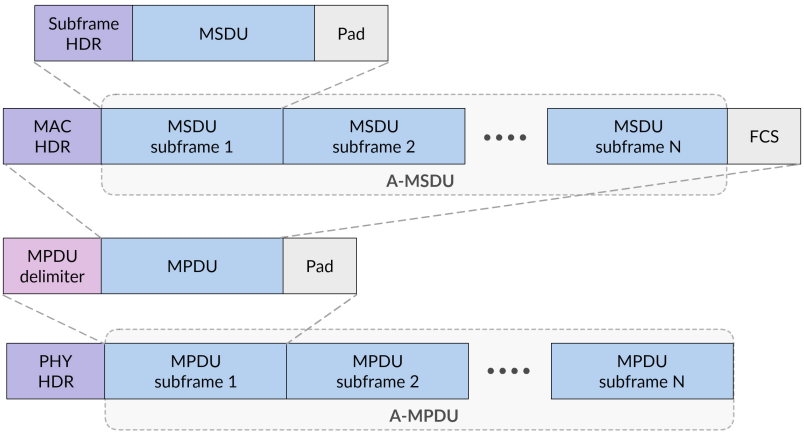


Figure 1: Frame aggregation in 802.11

2.3 MAC layer enhancements for WiFi

The Wireless Mesh Network used for performing the experiments presented in this paper is based on WiFi technology running IEEE 802.11 standards. WiFi technology has undergone an enormous evolution in the recent years. A recent measurement study [11] with millions of Cisco Meraki access points (APs) shows that around 99% of the APs use the 802.11n and 802.11ac standards. These standards introduce enhancements to increase data rates. Among them, *frame aggregation* is a simple method to enhance throughput, which we will show later to provide important opportunities to improve the performance of QUIC in WiFi scenarios.

The wireless medium has a high overhead, which includes the MAC and PHY headers, acknowledgments (ACK), backoff time and inter-frame spacing. For ACKs and small segments, the overhead in terms of bytes can be higher than the actual payload. The frame aggregation scheme amortizes this overhead and achieves high data throughput by combining multiple frames, also known as MAC Service Data Units (MSDUs), into a single transmission unit.

There are two main aggregation techniques in WiFi MAC architecture: Aggregate MAC Service Data Unit (A-MSDU) and Aggregate MAC Protocol Data Unit (A-MPDU) as shown in Fig. 1

A-MSDU: This technique receives and buffers multiple MSDUs destined to the same receiver (next station) and combines them into a single MPDU. The aggregation is completed when either the maximum A-MSDU size or the maximum delay is reached. The maximum A-MSDU size can be 3.8 k, 7.9 k or 11 k bytes and it is negotiated between the sender and receiver. The individual transmission sizes and delays vary between various driver implementations. The transmitter uses Cyclic Redundancy Check (CRC) for the subframes and MAC header to produce the Frame Check Sequence (FCS) which is used by the receiver to check for errors.

A-MPDU: This technique combines multiple MPDUs into a single unit having a common PHY header. There is no waiting/delay time in construction of A-MPDU and its size depends on the number of packets available in transmission queue. A-MPDU can have a maximum of 64 subframes and a maximum size of 65K bytes.

The performance gains of 802.11 frame aggregation have been extensively studied. Skordoulis et al. [12] provide a performance comparison of A-MSDU, A-MPDU and the combined two-level aggregation schemes and show that while all schemes improve throughput, each is suited for different environments. They point out that the efficiency of A-MPDU depends on the packet inter arrival rate because this scheme operates only on the packets that are already buffered in the transmission queue. Therefore, faster inter arrival rates result in higher throughput. Kim et al. [13] also evaluate the impact of frame aggregation on throughput. They show that, in general, increasing the frame aggregation size improves the overall throughput performance and reduces collision probability. They also show that faster frame arrival rates result in higher throughput. Lin et al. [14] provide a study of A-MSDU and A-MPDU frame aggregation schemes under error-prone channels using analytical and simulation methods.

As the 802.11 standard does not specify any particular frame aggregation scheduler, different vendors have implemented their own versions. Some prefer better resource utilization while others prefer timeliness and low delays. Various researchers have also proposed and evaluated different schedulers. Selvam et al. [15] present a scheduler that selects the aggregation size and technique (A-MSDU or two-level aggregation) based on the duration of frame transmit opportunity. Moh et al. [16] propose an algorithm that adjusts the aggregation size according to the frame error rate acceptable for the application. Hajlaoui et al. also propose a similar method where the scheduler considers specific QoS requirements for multimedia applications and dynamically adjusts the aggregated frame size. Saif et al. [17] propose an aggregation scheme (mA-MSDU) that reduces the aggregation headers and improves performance for applications that have a small frame size such as VoIP. Charfi et al. [**?**] show that 802.11n frame aggregation increases delay for low rate applications as the MAC layer has to wait for packets to construct the whole aggregated frame. They propose a new dynamic frame aggregation (DFA) scheduler to provide QoS satisfaction to real time services. Kim et al. [**?**] investigate the issue network throughput degradation due to fair allocation of channel access irrespective of packet size and data rate in 802.11n-based WLANs. They formulate an optimization problem for a generalized two-level frame aggregation and propose a frame size adaptation scheme that controls the number of packets in an aggregated frame. The perform simulations to show significant improvement in throughput as compared to other schemes. Coronado et al. [**?**] focus on software defined wireless networks and propose an adaptive ML-based approach for selecting frame size by relying on standard frame aggregation mechanisms. The proposed method obtains the required data from the control plane to adapt the frame size to the needs of each user. Experimental results show around 18% improvement in goodput over standard aggregation mechanisms.

While these prior studies study the issues of frame aggregation in WiFi networks by only focusing on the MAC layer, we instead investigate how the design decisions of transport protocols impact MAC layer features such as frame aggregation. In particular, we evaluate whether and how QUIC profits from WiFi frame aggregation.

2.4 Transport protocol enhancements for wireless networks

Given the popularity of TCP, several researchers have investigated different ways to improve its performance in wireless networks. A key factor affecting TCP performance in wireless networks is the contention and collision between ACK and data packets. Reducing the number of ACKs saves wireless resources and reduces interference with other packets thus improving performance. Moreover, lowering the ACK frequency increases the burstiness of traffic, as the sender releases a micro burst of packets after receiving the cumulative ACK. This behavior reduces the inter-packet time, increasing the opportunities for frame aggregation at the 802.11 MAC layer.

Altman el. al [18] investigated the impact of increasing the TCP delayed acknowledgement mechanism to more than two segments as recommended by RFC 1122. Singh et. al [19] propose TCP with adaptive delayed acknowledgement, which aims to reduce the number of ACKs to one per congestion window. Oliveira el. al [20] propose Dynamic Adaptive Acknowledgement where the delay window is adjusted according to the channel conditions. In [21], the same authors provide an improved delaying window strategy for robustness against losses.

Authors of [22] point out that the performance of TCP can degrade over wireless links due to spurious timeouts and spurious fast retransmissions caused by TCP's inability to distinguish between acknowledgments for original packets and retransmissions. The *Eifel scheme* uses TCP timestamps to avoid these issues. Authors of [23] study the sensitivity of TCP to sudden delay variations in mobile networks and quantify the risk of spurious TCP timeouts caused by changing RTT. Authors of [24] also report that long sudden delays during data transfers are not uncommon wireless WANs and can lead to unnecessary retransmissions and low throughput. They show that an aggressive TCP retransmission timer may trigger a chain of spurious retransmissions.

Similar research studies to understand the behavior of QUIC in wireless networks are largely missing. We provide a first study on the impact of different strategies on QUIC's performance in WiFi networks, focusing on WMNs.

2.5 State-of-the-art in QUIC performance evaluation

There are several recent research studies that compare the growth, security and performance of TCP and QUIC. Some explore the security of QUIC protocol [30, 31], while others study the implementation of multipath support [32, 33]. Rüth et al. [25] provide a broad assessment of QUIC usage in the wild. They monitor the entire IPv4 address space and about 46% of the DNS namespace to detected QUIC support. Their findings reveal that the number of QUIC-capable IPs has more than tripled between 2016 and 2017 and around 161 k domains are hosted on QUIC-enabled infrastructure.

Closer to our work, some research studies investigate the performance aspects of QUIC. Megyesi et al. [5] provide a comparative study of the performance of QUIC, SPDY and HTTP/1.1 in terms of Web page load time. They test QUIC v20 by hosting different pages on Google Sites, shaping the traffic between the client PC and Google Sites to emulate different bandwidth, RTT and packet loss rates. They show that with packet loss, SPDY performs the worst, followed by QUIC and HTTP/1.1. In case of high bandwidth and large page size, QUIC is three times faster than HTTP/1.1 and SPDY. Carlucci et al. [6] investigate performance of QUIC v21 on emulated networks using synthetic pages. They report that QUIC performs worse than HTTP/1.1, but better than SPDY with large web pages and 2% random packet loss rate. Without packet loss, QUIC performs better than HTTP/1.1 and SPDY for small and medium web pages, but worse for large pages due to its usage of only six parallel streams. Cook et al. [7] perform experiments in a local testbed, as well as using a 4G network card. They observe that QUIC and H2 have similar performance, but QUIC connections are less sensitive to delay and packet loss. In case of 4G links, QUIC outperforms H2. Kakhki et al. [8] evaluate QUIC v34 and TCP performance of web page loading and video streaming. They show that QUIC generally outperforms TCP, but there are some scenarios where QUIC's performance is reduced, e.g., high packet reordering in the network and large numbers of small objects in a web page. QUIC is however unfair with other protocols, taking more than 50% of the bottleneck bandwidth when competing with 2 or even 4 TCP connections. In our previous work [26, 27, 28] we analyze HTTP/1.1 and H2 traffic and evaluate performance of H2 over TCP and QUIC v39 under adverse network conditions using emulations.

In contrast to these studies which mostly focus on fixed networks, emulated network conditions with tools such as Linux tc and netem [29], or simplistic wireless network setups, this paper is focused on QUIC's performance in real-world WiFi networks. We perform experiments in a production wireless mesh network where the WiFi signals may be impaired by many factors, such as concrete walls, roofs or distance from access point. Additionally, the packets may travel through several wireless hops in the end-to-end path, increasing the chances of impairments in transit. Optimizing the protocols for wireless links can thus result in significant improvements in the application-layer performance. Our preliminary work in this direction is presented in [9]. In this paper we build upon our prior work and perform detailed analysis of congestion control and loss detection algorithms of QUIC protocol. We reveal the factors that impact the performance of QUIC in WiFi and optimize its performance.

3 Methodology

Our methodology to evaluate the performance of QUIC and TCP in WMNs is summarized in Fig. 2. We rely on a production WiFi mesh network, in which we deploy a series of client nodes. These nodes are connected to the Internet via multiple WiFi hops in the WMN. They download content from a remote Internet server using an instrumented Chromium browser. Logs are saved both at the client and server sides.

We next provide details about this setup, the way in which the server and clients are organized along with their specifications, and the performance metrics used in the experiments.

3.1 Wireless community network

We perform experiments in a production wireless community network deployed in a neighborhood of the city of Barcelona (Spain) called Sants [34]. The network was started in 2009 and in 2012 was joined by nodes installed at *Universitat Politècnica de Catalunya* (UPC) within the EU CONFINE

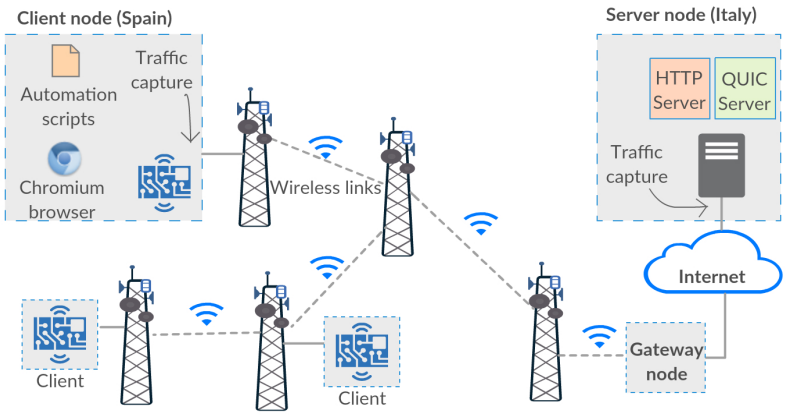


Figure 2: Experimental setup.

project [35]. The network is operative since 2009. The nodes use the linux/openwrt [36] based distribution provided by the Quick Mesh Project (QMP) [37], which runs the BMX6 mesh routing protocol [38]. From now on we will refer to this network as *QMPSU*. QMPSU is part of a larger community network started in 2004, which has more than 30.000 operative nodes called Guifi.net [39]. At the time of writing QMPSU has around 80 active nodes. Fig. 3 shows the geographic location of active nodes and links, using distinct colors to represent wireless links configured with different channels. In QMPSU there are 2 gateways that connect QMPSU to the rest of Guifi.net and the Internet.

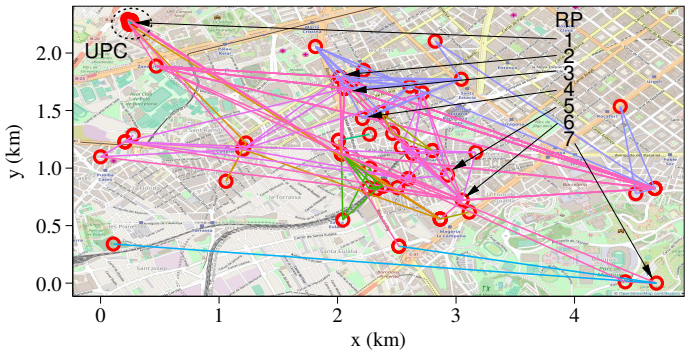


Figure 3: QMPSU topology. Colors indicate links configured in the same WiFi channel.

QMPSU is 802.11an-based and the most common hardware is the Ubiquiti NanoStation M5, equipped with a sectorial antenna and running QMP firmware[1]. There are also a number of point-to-point links using Ubiquiti parabolic antennas running the original manufacturer firmware. QMPSU also has a live monitoring web page updated hourly. A detailed description of QMPSU can be found in [40], and a live monitoring page updated hourly can be accessed on-line [41].

QMPSU has been deployed by its own users. Its unplanned spread out using heterogeneous WiFi devices in an urban area has produced a high diversity on the quality of the links. Thus, it offers a very realistic testbed to evaluate the performance of QUIC under a variety of conditions.

3.2 Experimental setup

We want to measure and compare the data transfer throughput achieved using TCP and QUIC between various clients in QMPSU and a server in the Internet. We deploy seven Raspberry PI 3 (RPi) devices attached to the premises of different volunteers across the QMPSU and use them as the client nodes. The RPis have quad-core ARM Cortex-A53 CPU and 1 GB RAM and run

[1]Note that some WiFi parameters are implementation-specific, such as the thresholds to perform frame aggregation.

Debian 9. Chromium browser v69 is used on the RPis in headless mode to load a web page from the server in the campus of Politecnico di Torino (Polito), Italy. The server has a dual-core Intel Core i5 CPU running Ubuntu 16.04.

The congestion controller in the server is TCP CUBIC. We use nginx web server for TCP and quic-go server supporting QUIC v43.

The clients in QMPSU network first reach the gateway node located in UPC using wireless links and then reach the server in Polito via the Internet as shown in Fig. 2. WiFi frame aggregation is enabled by default in all antennas in QMPSU. The geographical location of the clients is shown in Fig. 3, marked as *RP*. Tab. 1 shows the number of wireless hops (W-hops) in the WMN taken by the traffic downloaded from the server in Polito to reach the RPis. RP1 is located inside UPC and is directly connected to the fixed network without any WiFi links. Note that many of these WiFi hops use different frequencies and thus are not interfering with each other. Fig. 4 shows the average RTT between the clients and server and 95% confidence intervals[2]. The RTT was measured averaging over a large number of pings (more than 4,000 samples). The figure shows that RTT is in the range from 36 ms for RP1 to 90 ms for RP4, the client in the mesh having the weakest WiFi links.

Table 1: Characteristics of the client locations.

RP	Name	W-hops
*RP*1	UPC	0
*RP*2	BCCanBruixa20Rd6	1
*RP*3	BCNevaristoarnus5Rd3	2
*RP*4	BCNMelciorPalau62	5
*RP*5	GS26gener10	3
*RP*6	GSgV-rb	1
*RP*7	BCNJardiBotanicSants186	5

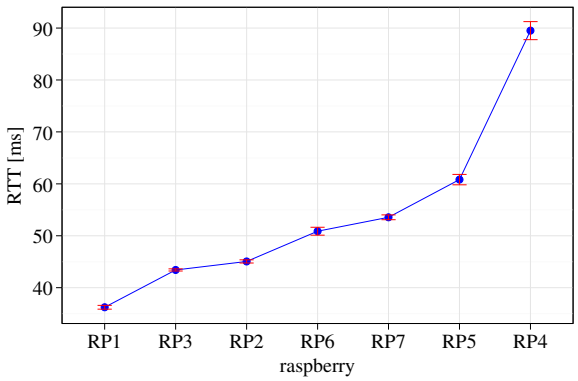


Figure 4: Mean and confidence interval of RTT between various clients and the server.

3.3 Measurements and performance metrics

We download a large image file of 10 MB hosted on our web server in Polito from the clients in Barcelona using TCP and QUIC. In both cases a single connection is used to load the web page. We cleanup client caches between repetitions of the experiments. Since the server is known after the first run of the experiments, for both TCP and QUIC we always go over the shortest steps to establish the encrypted connections (see Fig. 14). To automate the experiments, we deploy scripts that connect to remote clients in the mesh network hourly, to download the pages sequentially using every protocol under test. Traffic is captured at both client and server sides using tcpdump, and saved to a separate timestamped file.

We parse the captured traffic from each individual file and calculate various metrics such as the total page transfer time, throughput, and packet inter-arrival time etc. Throughput is computed

[2]95% confidence intervals of sample averages are computed using the Student's t distribution.

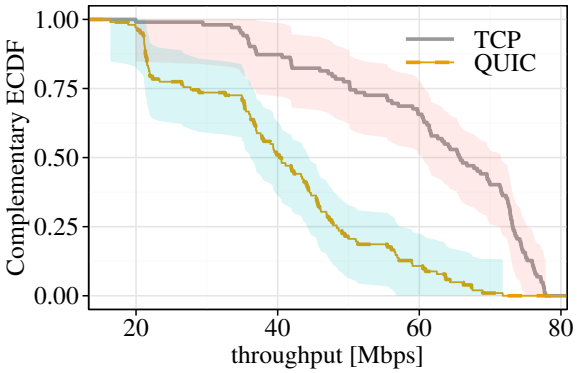


Figure 5: Throughput of TCP and QUIC for all experiments performed using RP3 client. The shaded areas are 95% confidence interval bands.

	Throughput [Mbps]			Loss prob. %	
	TCP	QUIC	TCP-Gain %	TCP	QUIC
RP1	71.9 ± 3.0	76.3 ± 0.33	-5.7	0.0054 ± 0.0073	0.039 ± 0.05
RP2	61.1 ± 3.0	40.9 ± 3.0	49.5	0.21 ± 0.094	0.52 ± 0.38
RP3	61.6 ± 2.9	40.3 ± 2.8	52.6	0.29 ± 0.15	0.29 ± 0.15
RP4	5.0 ± 0.46	4.5 ± 0.35	12.3	0.42 ± 0.058	0.45 ± 0.15
RP5	10.2 ± 0.72	7.9 ± 0.44	29.4	0.49 ± 0.088	0.32 ± 0.27
RP6	49.8 ± 2.4	38.2 ± 2.4	30.3	0.19 ± 0.064	0.15 ± 0.081
RP7	31.4 ± 2.1	21.6 ± 1.9	45.4	0.16 ± 0.096	0.15 ± 0.077

Table 2: Average throughput and loss rate of TCP and QUIC.

by dividing the number of bits sent in the payloads of the connection over the time of the transfer, ignoring connection establishment time. We also analyze the time sequence of data segments and ACK packets. We log TCP cwnd using tcpprobe kernel module.[3] In case of QUIC we instrument the web server to log the cwnd size on each ACK. Average values of these metrics are calculated using at least 200 experiments.

4 Evaluation

4.1 Performance comparison of TCP and QUIC

To evaluate the performance of TCP and QUIC we compare the average throughput of more than 200 experiments performed using each protocol as explained in Section 3.3.

WiFi frame aggregation is enabled by default in all antennas and both TCP and QUIC use a single connection between the client and the server to load the web page. Tab. 2 shows the mean throughput and packet loss probability of TCP and QUIC for various clients. RP1 is the client located at UPC and is directly connected to LAN *without* any WiFi link. QUIC achieves slightly higher throughput of around 76 Mbps as compared to TCP's 72 Mbps on this wired link (5.7% gain). This result conforms to prior studies that also find QUIC to perform slightly better than TCP. However, the outcome is very different with other clients that involve one or more WiFi hops on their end-to-end path. We can see that in all cases TCP achieves higher throughput than QUIC. The difference is much higher is high-bandwidth links – e.g., in case of RP2 and RP3 TCP's throughput is larger than QUIC by roughly 50%. This TCP-gain is around 12%, 29%, 30% and 45% for RP4, RP5, RP6 and RP7 respectively as shown in Tab. 2. We can also see that the packet loss rate is the range of 0.15% to 0.52% for various clients connecting through WiFi links. The loss rate is almost zero for the client RP1 connected through wired link.

[3] https://wiki.linuxfoundation.org/networking/tcpprobe

Table 3: Comparison of QUIC and TCP performance under stress.

	TCP Throughput (Mbps)			QUIC Throughput (Mbps)		
Device	Normal	Stressed	Decrease	Normal	Stressed	Decrease
RP2	58.5	53.47	8.6%	38.8	17.7	54.3%
RP7	31.8	29.6	6.9%	21.2	16.6	21.6%
RP5	10.2	9.89	3%	7.3	6.1	16.4%

Fig. 5 shows the complementary ECDF of the throughput achieved using all the experiments performed using one of the clients, i.e., RP3. For each curve Fig. 5 shows 95% confidence intervals bands[4]. We can see that TCP achieves higher than 60 Mbps throughput in more than 66% of experiments, while this number is only 11% for QUIC. On the other hand, TCP achieves less than 30 Mbps throughput in only 2% cases while this number is 26% for QUIC.

4.2 Impact of the userspace implementation of QUIC on results

TCP is implemented in the OS kernel while QUIC is implemented in userspace. This could have negative impact on performance, particularly on resource-constrained mobile devices. This impact was first shown experimentally by Kakhki et al. [8]. They showed that on a mobile phone (MotoG) QUIC v34 spends 58.84% of the time in "application limited state" as compared to only 7.05% on a PC. In application limited state the cwnd is not increased any further because the client may be slow and unable to process the incoming packets at the sender's rate.

Since we use RPis to perform experiments, we check to what extent the userspace implementation of QUIC in RPis impacts our results. To this end, we instrument the source code and log the state machine of QUIC during experiments.

In a first experiment, we load the same web page with a 10 MB image from the server over QUIC using two clients, an RPi and a PC, both located at the same place. Both devices have no other client processes running during the experiments. This experiment is repeated 30 times and the mean time expended in which QUIC state are calculated. Results of this experiment are summarized in Fig. 6. There is little difference between RPi and PC in the amount of time spent in the various states. This shows that the performance of QUIC has improved over time (when compared to [8]), and its userspace implementation is not a limiting factor on moderately powerful devices such as RPis. It also validates our choice of using RPi as client.

Yet, this first experiment has been performed without any other load on the devices. We also check whether performance would be affected in case clients have high workloads – e.g., due to other process running simultaneously. We thus repeat the experiment while stressing the CPU of some RPis using *Linux stress tool*. We record mean throughput and compute the difference between the stressed and unstressed setting. We perform this experiment on three RPis which have different link quality and capacity and the results are shown in Table 3.

We can see that stressing the CPU has a big impact on QUIC throughput which drops by up to 54%. TCP throughput is reduced by a maximum of 8.6% in the worst case. We also observe that the higher the bandwidth the bigger the reduction is. In low bandwidth links the network becomes the bottleneck therefore, stressing the CPU has little impact.

These results show that QUIC's userspace implementation can indeed have negative impact on low-end devices with limited processing power. However, with moderately powerful devices this is not an issue. For the experiments presented in this paper, we carefully avoid any other load on the client devices, since our focus is to understand the performance of the protocols in similar testing conditions.

[4]95% confidence bands of ECDF have been estimated via the Kolmogrov-Smirnoff approach using R package [**?**].

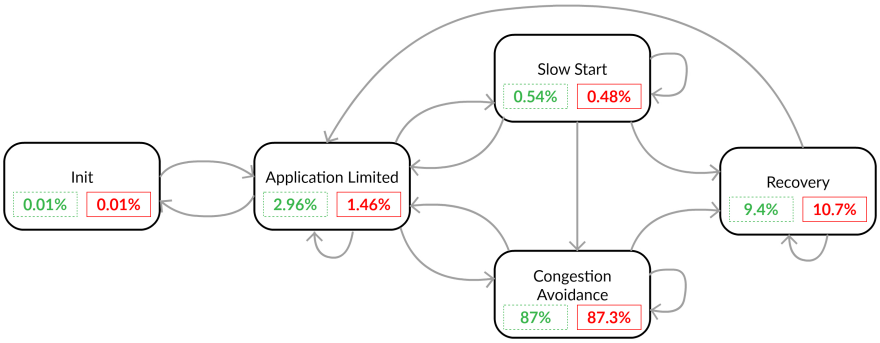


Figure 6: QUIC v43 state transitions on RPi (green dotted box) vs PC (red solid box). The numbers show the percentage of time spent in each state. The behavior is very similar on both devices.

Table 4: List of all input features.

No.	Features	Details
1	Server/client mean inter-packet time	The inter-departure time between packets on the server side and the inter-arrival time between packets on the client side
2	Server/client inter-packet time variation	The variance of inter-packet time of server/client
3	Server/client burstiness	The burstiness is computed using the index of dispersion for intervals as described in [42]
4	Mean delay	The mean value of one way delay of all data packets transferred from server to client
5	Delay variation	The variance of one way delay
6	Packet loss	Probability of packet lost. It is defined as the fraction of the total transmitted packets that did not arrive at the receiver, calculated by dividing the number of lost packets over the total transmitted packets
7	Out of order	Probability of out of order packets. It is defined as the fraction of the total transmitted packets that are received in a different order from which they were sent
8	ACK freq	ACK frequency is the ratio of acknowledgements and data packets
9	Mean cwnd	Mean value of the congestion window measured throughout the data transfer
10	Cwnd variation	Variance in the size of congestion window
11	Slow start/ Congestion avoidance/ App limited/ Recovery freq	A vector containing 4 features: The frequency of slow start, congestion avoidance, recovery and application limited phase respectively. This is the percentage of time spent in a particular phase during the data transfer

5 Determining factors of TCP and QUIC performance

To identify the determining factors of TCP and QUIC performance in WiFi, we follow a two-step process. First, we apply a "black-box approach", using machine learning techniques, to identify key factors that impact the throughput of the protocols. Once we have identified these factors, in the second step we inspect QUIC source code to further investigate them to understand the root-causes of the performance issues.

We analyze the network traffic captured at the client and server side and compute several attributes/features from individual network connections. These features are derived from information in IP and TCP/UDP headers and logs of cwnd. The protocol design as well as the network conditions may dictate the value of a feature. Table 4 shows the list of all features computed for each individual connection. With this list, we apply classic *feature selection* methods to assess how

they correlate with the throughput of connections. Then we present the *subset* of features that influences throughput the most.

5.1 Subset feature selection

To see how various combinations of features impact throughput, we use a decision tree to learn how to predict the throughput of connections from the given features. More than a precise method to predict throughput, we are interested in understanding which subset of features contributes the most for the predictions. Decision trees are simple to understand and interpret and provide useful information about the relationship between various features.

Our target variable is the throughput. For the sake of simplicity, we divide throughput into discrete bins of 10 Mbps that are classified as *t0*, *t10*, *t20*, *t30* etc, where class *t0* represents throughput in the range of 0 Mbps to 10 Mbps, class *t10* represents throughput in the range of 10 Mbps to 20 Mbps and so on. We take roughly 2 000 network connections from our experiments. For each network connection we extract all features, assign a class based on its throughput range, and then apply decision tree learning on the dataset. We use the implementation of the C4.5 decision tree algorithm offered by Weka [43].

The resulting decision trees for TCP and QUIC are shown in Figure 7a and 7b. The leaf nodes of the tree depict the throughput class while the intermediate nodes show the factors that result in the corresponding throughput. The value next to each leaf node is the percentage of accurately classified instances of that class. The overall accuracy, precision and recall is 80.05%, 79.8% and 80.01% respectively for QUIC and 81%, 81.3% and 81% respectively for TCP. The trees confirm that factors impacting throughput are different for TCP and QUIC. While cwnd is the most important factor for both protocols, slow throughput is associated with packet losses, server burstiness, and mean delay, in the case of TCP, but with slow start frequency, delay variation and client burstiness, in the case of QUIC.

These results are partly expected and can be explained with domain knowledge and intuition. For example, the cwnd defines how much unacknowledged data can be in-flight before an ACK is received. Generally, the larger the cwnd size, the higher the throughput. Both feature selection approaches capture this intuition and show how the cwnd setting is vital for both QUIC and TCP.

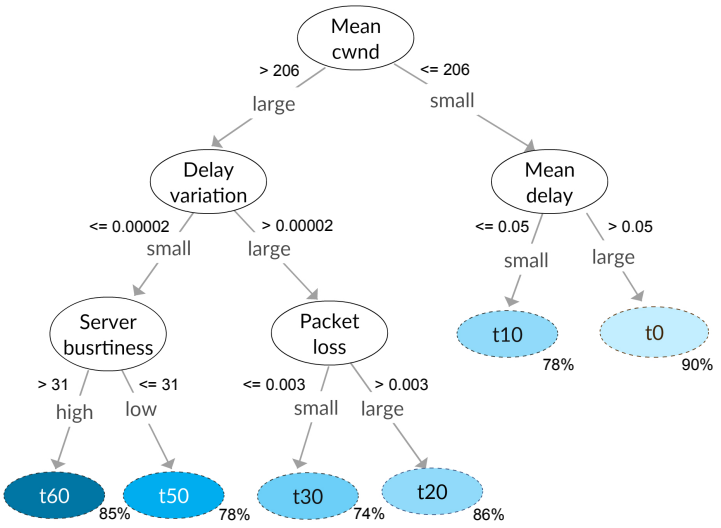
Instead, the difference on importance of other features is not immediately obvious and call for further analysis. For example, the algorithms that control the slow start, congestion avoidance etc. specify how to grow the cwnd size. The slow start phase is designed to quickly converge on the available bandwidth between the client and the server. During this phase the cwnd increases exponentially on each RTT. Once we are out of the slow start and into the congestion avoidance phase the cwnd grows at a much lower rate. The slow start phase seems to affect QUIC more than TCP. Similarly, the end-to-end delay impacts the growth of cwnd because the sender has to wait for ACKs from the receiver before increasing the sending rate. Wireless connections are particularly susceptible to signal interference that causes network data to be corrupted in transit, increasing delays due to link layer retransmissions. In the same way, packet losses impact the sending rate in a similar manner. TCP and QUIC are both reliable transport protocols, therefore they react to packet losses by reducing the sending rate, albeit to different extent, and retransmitting the lost packets.

In the next section we further investigate the factors identified by the decision trees by digging into the source code of QUIC v43. We try to find out why these factors impact QUIC and TCP protocols differently by critically evaluating the design decisions of QUIC.

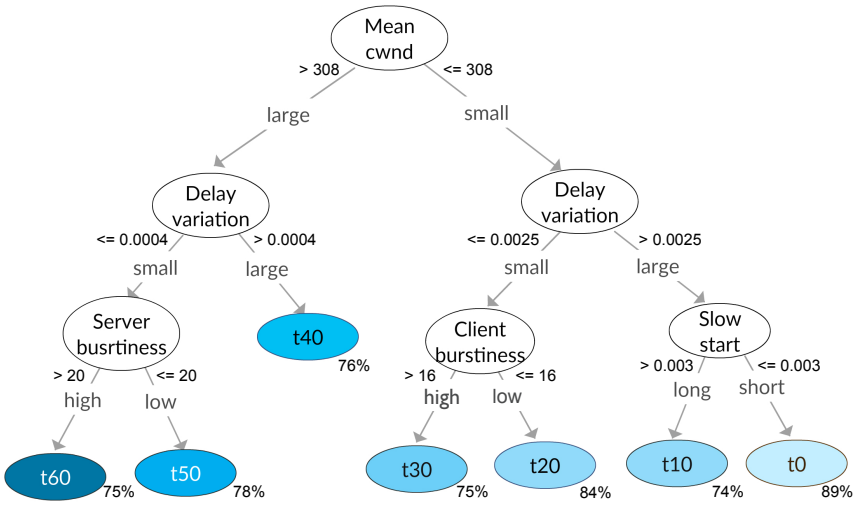
5.2 Root-causes of performance issues of QUIC in WiFi

The results of our experiments in Sec 4 show that QUIC has lower performance than TCP in the WMN. The decision trees provide useful information and show that cwnd size, high delay variation, client burstiness etc. impact QUIC's performance. We now further investigate these factors to identify the root-causes for the problem.

Since the cwnd size dictates throughput, we start by analyzing the cwnd of TCP and QUIC, instrumenting Chromium to log their values during all our experiments. As a reference, we select RP3, the client with the highest quality WiFi links, and measure the available bandwidth capacity



(a) Decision tree for TCP.



(b) Learned decision tree for QUIC.

Figure 7: Simplified decision trees for TCP and QUIC show how various features result in throughput shown at the leaf nodes. Threshold for splitting the nodes is shown at the branches. The percentage of accurately classified instances is shown next to each leaf node.

and the RTT between the client and the server using netperf and ping, respectively. We estimate the *optimal* cwnd size (minimal window allowing the maximum throughput) using the well-known bandwidth-delay product:

$$Optimal\ cwnd = Bandwidth \times RTT$$

For RP3, with 62 Mbps of available bandwidth and 44 ms RTT, the cwnd size needs to be at least 340 kB to keep the link saturated. Figure 8 shows the complementary ECDF of the mean cwnd size measured during more than 200 experiments each for TCP and QUIC. For each curve Fig. 8 shows 95% confidence intervals bands. The blue vertical line represents the optimal cwnd size. We can see that more than 71% of cases TCP finishes the transfer with a mean cwnd size larger that the optimal value, while this occurs only in 30% of the cases with QUIC. Furthermore, in around 25% of the cases the cwnd size in QUIC is smaller than half of the optimal size.

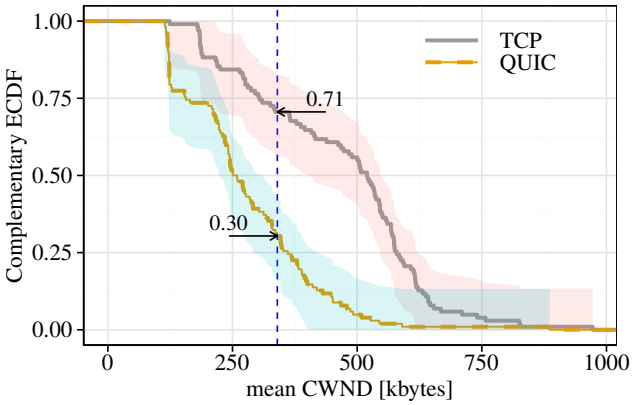


Figure 8: Mean cwnd comparison for RP3 client. The blue vertical line is the optimal cwnd size calculated using the bandwidth delay product. The shaded areas are 95% confidence interval bands.

Table 5: Frequency of normal and abnormal behavior of QUIC observed during the measurement campaign.

Early exit	False loss detection	Normal
39%	51%	10%

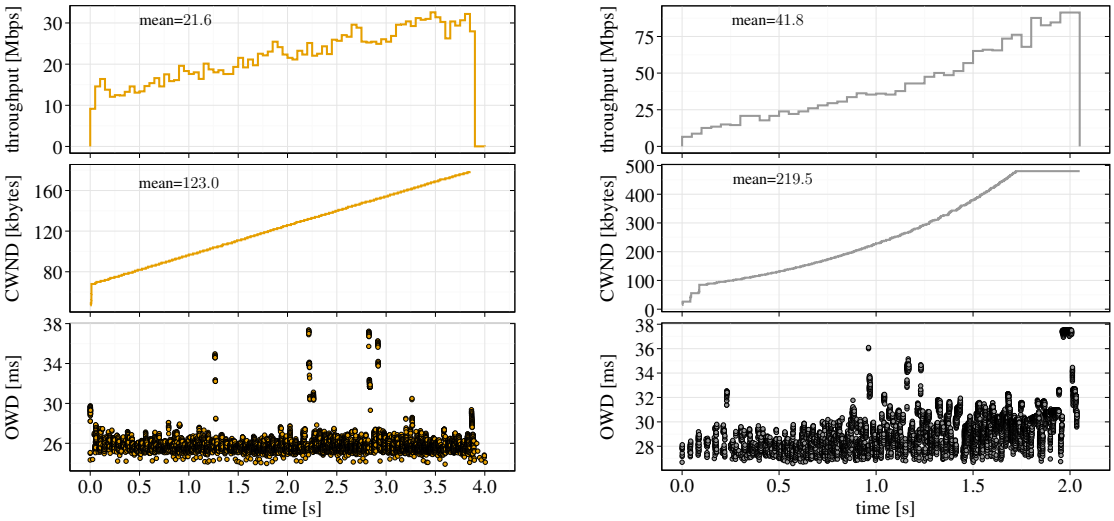
To see why QUIC selects sub-optimal cwnd so often, we plot individual cwnd charts for over 200 experiments performed using QUIC and TCP. We observe that two abnormal behaviors occur in QUIC quite frequently. First, QUIC often exists early from slow start, before reaching the optimal cwnd size, and while in the congestion avoidance phase it increases the cwnd at a slow pace. Second, QUIC spuriously detects congestion and reacts by reducing cwnd to a sub-optimal size. In TCP such a behavior is not very common. Early exit from slow start in TCP is usually due to an actual packet loss and false loss detection due to packet re-ordering which generates multiple duplicate ACKs. The causes for this behavior in QUIC are explained in detail next. Table 5 shows the frequency of normal and abnormal behavior of QUIC during experiments.

Cause 1: QUIC v43 exits early from slow start and has a sluggish congestion avoidance phase

In slow start phase the cwnd size is increased exponentially. Premature exit from slow start results in underestimation of bandwidth and low throughput. As explained in Sec 2.2, QUIC v43 congestion control algorithm exits from slow start phase when an increasing delay is detected (RTT has increased by more than $1/8^{\text{th}}$). E.g., if minimum RTT is 40ms, an increase of up to 5ms from this value can be tolerated for the slow start phase to continue. This idea works well in wired networks where RTTs are quite stable and a significant increase indicates congestion. However, the same does not hold true for WiFi networks where it is common to have variation in RTT due to factors related to the underlying medium and not due to network congestion.

High delay variation in the network at the start of a transfer forces QUIC to exit early from slow start phase. As a result, the cwnd cannot increase to the optimal size and leads to low throughput. This behavior of QUIC is shown in Fig. 9a which compares the delay, cwnd size and throughput of QUIC for a single transfer using RP3 where this behavior occurs. We can see that right at the start of the transfer the delay varies between 24 ms and 30 ms. This variation of around 6 ms is larger than QUIC's threshold of 3 ms which results in early exit from slow start.

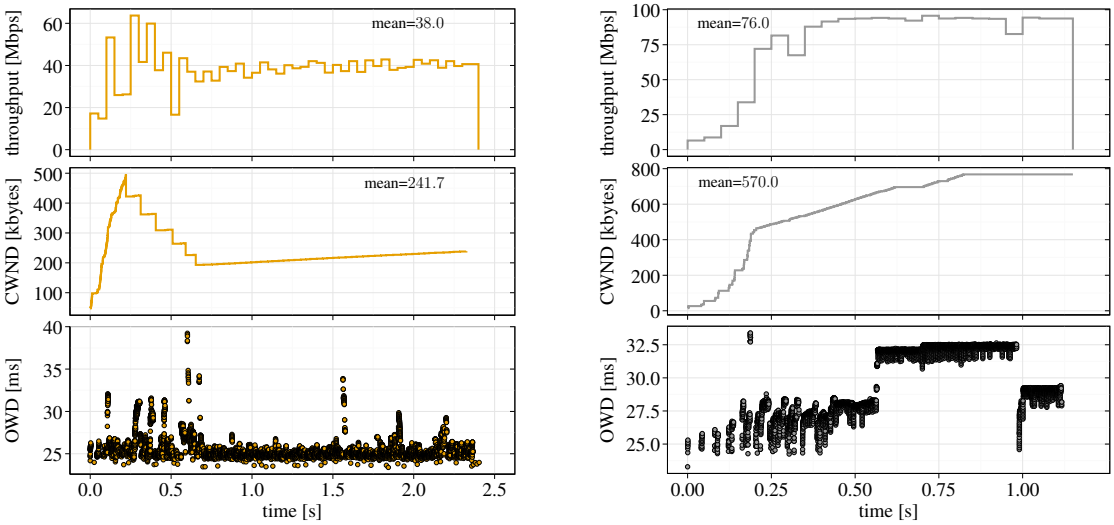
Although TCP also exits early from slow start occasionally, as shown in Fig. 9b, the impact of this behavior is much more damaging for QUIC because after exiting early from slow start QUIC's cwnd increases very slowly in congestion avoidance phase while this rate is much faster in case of TCP. This can be observed in cwnd chart in Fig. 9a where QUIC's cwnd increases linearly and it takes around 4 s to increase the cwnd from 70 kB to 180 kB. QUIC is not able to reach the optimal cwnd size of 320 kB even until the end of the transfer. On the other hand, TCP has



(a) QUIC v43 early exit from slow start.

(b) TCP's early exit from slow start.

Figure 9: High delay variation affects QUIC v43 slow start and congestion avoidance algorithms.



(a) QUIC spuriously detects losses

(b) Normal TCP download

Figure 10: High delay variation affects QUIC's loss detection algorithm

a parabolic curve and it takes only around 1.6 s to increases cwnd from 90 kB to 480 kB. The corresponding mean throughput of this download is around 21.6 Mbps for QUIC and 41.8 Mbps for TCP on the same link. This behavior can be attributed to the slow packet pacing rate used by QUIC v43 when not in the slow start phase. This results in under-utilization of the available cwnd. We have measured the percentage of experiments when QUIC and TCP exit early from slow start and are not able to reach the optimal cwnd size before the end of the download. This is value only 8.5% for TCP and 100% for QUIC. In other words, QUIC never attains the optimal cwnd size in any of our experiments when it exits early from slow start.

Cause 2: QUIC v43 spuriously detects congestion

Spuriously detecting congestion and declaring packets as lost causes unnecessary retransmissions and may result in performance degradation because the protocol takes action to reduce the sending rate.

Table 6: Throughput achieved over a WiFi link with frame aggregation enabled and disabled using RP6 client.

	Throughput (Mbps)		
	Frame aggregation disabled	Frame aggregation enabled	Gain(%)
TCP	26.1 ± 1.3	49.8 ± 2.4	90
QUIC	23.7 ± 1.1	38.2 ± 2.4	61

As explained in Sec 2.2, QUIC v43 loss detection algorithm tracks the unacknowledged packets and declares those packets as lost whose acknowledgement is not received within a fixed threshold of $1/8^{\text{th}}$ of the RTT from the packet sending time. This threshold seems quite conservative and causes performance degradation in WiFi networks with high RTT variation.

Fig. 10a shows this particular behavior during a download when QUIC stays in slow start for longer duration and is able to grow its cwnd to a good size, but soon afterwards it starts reducing the cwnd repeatedly, and we see a staircase pattern going downwards. This is again caused by high delay variations that are visible in one-way-delay (OWD) chart[5]. This reduction in cwnd size is triggered by QUIC v43 loss detection algorithm. However, using the IP identification field from the client and server side, we confirm that there is *zero* packet loss during this transfer. As a result, the mean throughput of QUIC is limited to 38 Mbps. We rarely observe this with TCP. Our experiments shows that this behavior occurs in almost 51% of cases for QUIC and only 1.9% for TCP. Fig. 10b shows the normal behavior of TCP download on the same client and link.

Cause 3: QUIC v43 does not exploit advanced WiFi transmission methods

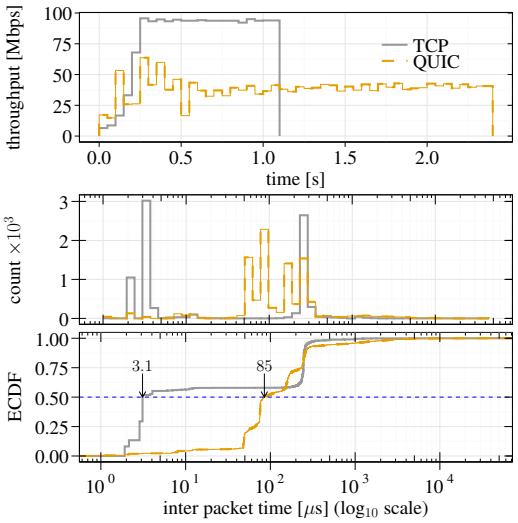
QUIC aims to reduce traffic burstiness and, as a consequence, queuing delays and packet losses by using *packet pacing* [44]. We observe that due to the characteristics of the WiFi medium and interactions between transport and MAC protocols, bursty traffic might actually be beneficial in WiFi. One apparent reason for this behavior is frame aggregation. As mentioned in Section 2.3, frame aggregation in recent 802.11 standards is a key feature to achieve high throughput, and bursty traffic with short inter-packet time increases frame aggregation opportunities [12][13].

In order to measure the impact of frame aggregation on TCP and QUIC performance we repeat our experiments by disabling frame aggregation in the antenna located at UPC. The traffic from the client RP6 in the WMN passes through this WiFi link to reach the gateway and then the server in Polito. In order to assess the relevance of frame aggregation and to observe how both protocols exploit it, Table 6 shows the throughput and 95% confidence intervals obtained for RP6 with and without frame aggregation in the WiFi link. The table shows that with frame aggregation the throughput is increased by almost 90% with TCP and only 61% with QUIC. This shows that TCP can efficiently exploit frame aggregation while QUIC is not able to fully benefit from it.

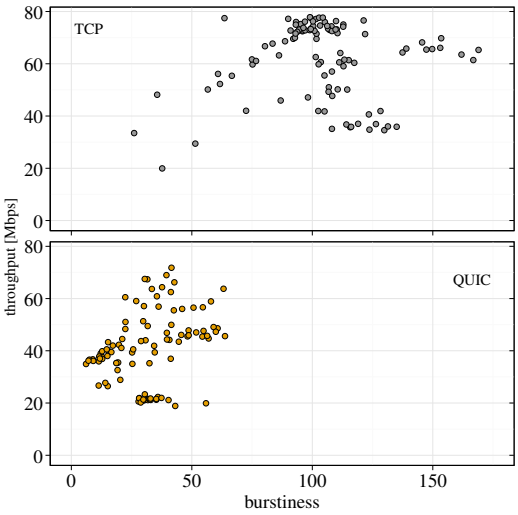
To see why this is the case, we compare the inter-packet time of both protocols. As an example, Fig. 11a shows the ECDF and histogram of the inter-packet time of TCP and QUIC for one of the downloads from RP3. This information is retrieved from the server side packet traces of two back-to-back experiments done using TCP and QUIC. We can see that around 50% of TCP packets have inter-packet time between 2μs and 4μs while for QUIC this range is between 50μs and 100μs. Large inter-packet time means less chances of exploiting frame aggregation.

Figure 11b shows a scatter plot of the throughput vs. burstiness for all downloads from RP3. For TCP the figure shows a clear correlation between throughput and burstiness: the higher is the burstiness the higher is the throughput. However, burstiness is not the only parameter influencing the throughput. This explains why there is a small proportion of points having a relatively low throughput even if the burstiness is high. As expected, Figure 11b shows that QUIC is much less bursty than TCP, due to packet pacing.

[5]OWD is measured as the timestamp difference of the corresponding packets in the tcpdump traces captured at the server and client sides.



(a) Inter-packet time comparison of TCP and QUIC at the sender.



(b) Impact of burstiness on throughput of TCP and QUIC.

Figure 11: TCP gets higher benefits from WiFi frame aggregation due to its bursty traffic

6 Optimizing QUIC v43

6.1 Modifications

Taking into consideration the performance issues of QUIC v43 in WiFi, we incorporate the following modifications in QUIC source code: (1) We use `ACK_DECIMATION` as default acknowledgment mode which reduces the transport layer acknowledgment frequency, increasing traffic burstiness at the server side and thus resulting in greater aggregation opportunities at the WiFi MAC layer. (2) We make QUIC's congestion controller more tolerant to RTT variation in WiFi networks by increasing the threshold defined for exiting slow start. (3) We make QUIC's time-based loss detection algorithm more resilient in WiFi networks by increasing the threshold defined for considering a packet lost.

Since these features are not controllable from the browser configuration, we had to study the source code, make required modifications, and cross compile a customized version of Chromium browser for ARM architecture and deploy it on the RPis. We call this version *Bursty QUIC* (BQUIC).

We have carefully proposed these changes in the QUIC protocol design so that tweaking it to make it perform better on one particular link layer does not have negative effects in the general case. We validate the effectiveness of these changes by performing experiments in our production WMN testbed where the end-to-end path between client and server contains one or more WiFi last hops as well as wired links in the Internet.

6.2 Performance evaluation of QUIC v43 and BQUIC

To better understand the difference in behavior of QUIC and BQUIC and their performance, we compare various parameter including throughput, loss rate and one-way-delay (OWD). Tab. 7 presents the mean values of all experiments for each client with 95% confidence intervals.

We can see that BQUIC achieves much higher throughput. The gain in throughput ranges between 7% to 28% for different clients. Clients with high performance WiFi links generally show higher gains and vice versa. One of the reasons is that bad quality links are usually limited by factors such as noise or interference etc., that cause packet losses.

The WiFi MAC layer retransmits lost unicast frames multiple times before abandoning its transmission which increase end-to-end delay that can be observed in case of RP4 and RP5 in Tab. 7. The bottleneck is such cases is the bad link quality therefore the proposed optimizations, namely improving slow start and loss detection algorithms and exploiting WiFi frame aggregation

have little benefits. In high quality links the QUIC protocol design factors act as bottleneck therefore and optimizing it results in significant improvement in performance which can be observed in case of RP2 and RP3. The client RP1 that is connected via fixed network shows very little performance improvement since our proposed optimizations particularly target issues of QUIC in the WiFi medium. However, the case of RP1 is very important as it shows that these optimizations do not have negative effects in fixed networks.

Table 7: Performance comparison of QUIC and BQUIC

Device	Throughput (Mbps)			Loss prob. (%)		OWD (ms)	
	QUIC	BQUIC	Gain	QUIC	BQUIC	QUIC	BQUIC
RP1	76.3 ± 0.33	80.4 ± 0.98	5.4%	0.039 ± 0.05	0.011 ± 0.012	12.6 ± 0.86	13.2 ± 1.0
RP2	40.9 ± 3.0	50.6 ± 4.0	23.8%	0.52 ± 0.38	0.32 ± 0.19	22.0 ± 0.75	22.8 ± 0.76
RP3	40.3 ± 2.8	51.7 ± 3.9	28%	0.29 ± 0.15	0.26 ± 0.12	25.6 ± 0.37	26.3 ± 0.43
RP4	4.5 ± 0.35	4.8 ± 0.37	7.3%	0.45 ± 0.15	0.39 ± 0.085	93.1 ± 4.4	86.6 ± 4.6
RP5	7.9 ± 0.44	9.2 ± 0.71	16.3%	0.32 ± 0.27	0.29 ± 0.074	56.7 ± 8.6	58.5 ± 8.7
RP6	38.2 ± 2.4	42.3 ± 3.6	10.7%	0.15 ± 0.081	0.33 ± 0.2	22.2 ± 2.5	23.8 ± 2.5
RP7	21.6 ± 1.9	26.1 ± 2.6	20.5%	0.15 ± 0.077	0.17 ± 0.094	41.5 ± 3.1	47.0 ± 4.7

Next, we compare the loss rate of QUIC and BQUIC, which have been computed by comparing the identification field of the IP header of transmitted and received datagrams. In most cases the loss rate is slightly lower in BQUIC. An apparent reason for this is the higher rate of frame aggregation in BQUIC which combines more packets into a single unit and transmits it over the WiFi link, thus reducing the probability of loss of individual packets. It also validates that making QUIC traffic slightly more bursty does not increase packet losses, which is commonly expected. In case of RP6 and RP7 we do observe higher loss rate but the increase in minimal. Lastly, in case of RP1 which we see almost no packet losses which is common for fixed networks.

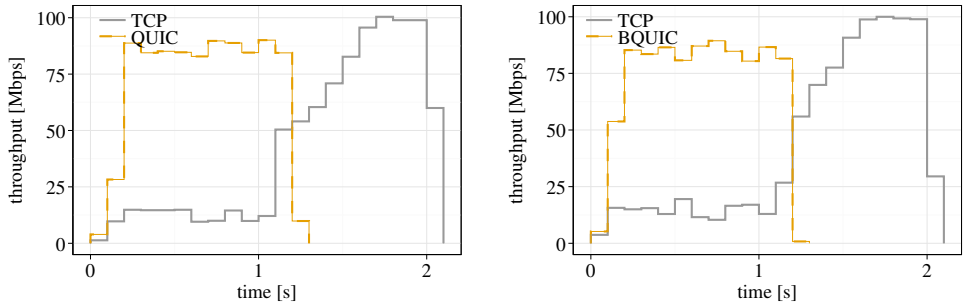
We also compare the mean one way delay of the packets to see if the increased traffic burstiness in BQUIC causes queues in the middleboxes to build up, which would result in increase in delay. In most of the good quality links the delay is almost the same for QUIC and BQUIC. As explained earlier, with BQUIC we observe lower packet losses due to higher frame aggregation in case of bad quality links that are highly prone to errors. Lower loss rate means less MAC layer retransmissions which result in lower delays. This can be observed for RP4 which has the worst quality links and the delay is reduced from 93.1 ms to 86.6 ms.

6.3 Fairness comparison

Fairness among the competing flows of various transport protocols is very important for a network. It has been shown in prior studies that QUIC consumes greater share of bottleneck bandwidth as compared to TCP. In our previous work [28] we observed in an emulated environment that a QUIC v39 flow consumes around than 70% of the bottleneck bandwidth when competing with a single TCP flow and around 50% when competing with 2 TCP flows. [8] have shown even greater unfairness with QUIC v34 where QUIC consumes more than half of the bottleneck bandwidth even when competing with 4 TCP flows.

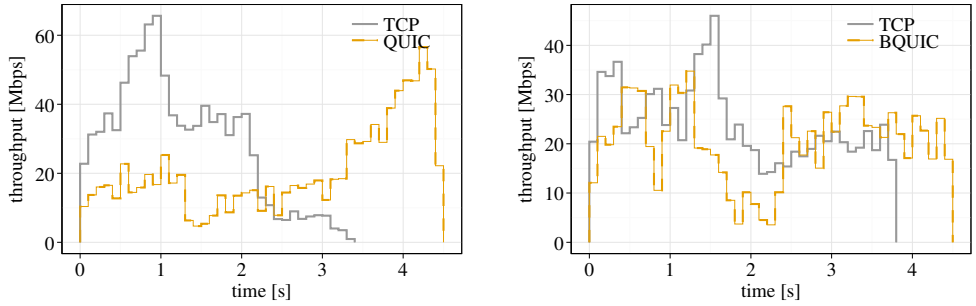
Here our main objective is to measure the fairness of QUIC v43 and BQUIC with TCP in a WiFi network. We have shown that BQUIC is able to achieve higher throughput than QUIC in WiFi but it is crucial to make sure that it does not starve other competing flows. We perform measurements using two different links. First is the wired link between the two university campuses in Spain and Italy. Second is the WiFi link between a client in the mesh network and UPC campus. We run a script that starts two parallel threads on the client to fetch a web page from the server using TCP and QUIC. We repeat the experiment using TCP and BQUIC.

Fig. 12 shows the results of the wired network. In agreement with prior studies, we can see that QUIC consumes much higher share of the bottleneck bandwidth than TCP. BQUIC also has similar behavior and we do not find it to be more aggressive than QUIC, confirming that our proposed changes do not have a negative impact in fixed networks.



(a) TCP vs QUIC (b) TCP vs BQUIC

Figure 12: Fairness comparison in wired network



(a) TCP vs QUIC (b) TCP vs BQUIC

Figure 13: Fairness comparison in WiFi network

In case of WiFi network the behavior of QUIC and TCP is very different as shown in Fig. 13. Here TCP gets much higher share of bandwidth and QUIC is able to grab more bandwidth only after the TCP flow is about to finish. Interestingly, the competing flows of TCP and BQUIC are quite fair to each other which is a positive thing. Besides providing higher throughput, our proposed optimization also provides better fairness with competing TCP flows in WiFi.

7 Conclusions

In this paper we evaluated the performance of QUIC v43 in WMN. We first highlighted that while QUIC achieves better throughput than TCP in wired networks, it achieves sub-optimal throughput in high performance WiFi links. We showed this by performing experiments in a production WMN. Using machine learning techniques, we determined the factors that impact TCP and QUIC performance and then proceeded to investigate the root-causes of QUIC's performance degradation in the WMN. We discovered that high variation in delay in WiFi links can be misinterpreted by QUIC as a sign of congestion and it can also result in early exit from slow start phase. Another cause is the design decision to make QUIC traffic smooth by using packet pacing, which does not allow it to efficiently exploit 802.11 frame aggregation. We implemented and evaluated *BQUIC*, i.e., a customized version of QUIC v43 that increases traffic burstiness to better exploit WiFi frame aggregation and optimizes threshold for slow start and loss detection algorithms. Our results show that BQUIC achieves up to 28% higher throughput than vanilla QUIC.

While our experiments focus on WMNs, we believe that the findings may be applicable to other WiFi networks as well. The reason is that the root-causes of the performance issues are related to the transport protocol design and its interaction with the WiFi MAC layer. This work opens opportunities for other researchers to study cross-layer optimizations. Layer-2 protocols in wired and wireless networks are nowadays significantly different, this, coupled with the flexibility of modification provided by QUIC opens the opportunity for researchers to study cross-layer optimizations and dynamically tune the protocol for each network type. This can greatly improve the overall performance and end-user experience.

Acknowledgments

This work was supported by the Erasmus Mundus Joint Doctorate in Distributed Computing EMJD-DC program, the Spanish grant TIN2016-77836-C2-2-R, and Generalitat de Catalunya through 2017-SGR-990. This research was conducted as part of the PhD thesis which is available online at upcommons.upc.edu.

References

[1] Leonardo Maccari and Renato Lo Cigno. A week in the life of three large wireless community networks. *Ad Hoc Networks*, 24:175–190, 2015.

[2] Cisco Visual Networking Index. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html`, 2018. (accessed May 2019).

[3] Guido R Hiertz, Dee Denteneer, Lothar Stibor, Yunpeng Zang, Xavier Pérez Costa, and Bernhard Walke. The ieee 802.11 universe. *IEEE Communications Magazine*, 48(1), 2010.

[4] Adam Langley et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the SIGCOMM*, pages 183–196, 2017.

[5] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is quic? In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[6] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Http over udp: an experimental investigation of quic. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 609–614. ACM, 2015.

[7] Sarah Cook, Bertrand Mathieu, Patrick Truong, and Isabelle Hamchaoui. Quic: Better for what and for whom? In *IEEE International Conference on Communications (ICC2017)*, 2017.

[8] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*, pages 290–303. ACM, 2017.

[9] Jawad Manzoor, Llorenç Cerdà-Alabern, Ramin Sadre, and Idilio Drago. Improving performance of quic in wifi. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019.

[10] QUIC Implementations. `https://github.com/quicwg/base-drafts/wiki/Implementations`. (accessed May 2019).

[11] Apurv Bhartia, Bo Chen, Feng Wang, Derrick Pallas, Raluca Musaloiu-E, Ted Tsung-Te Lai, and Hao Ma. Measurement-based, practical techniques to improve 802.11 ac performance. In *Proceedings of the 2017 Internet Measurement Conference*, pages 205–219. ACM, 2017.

[12] D. Skordoulis, Q. Ni, H. h. Chen, A. P. Stephens, C. Liu, and A. Jamalipour. Ieee 802.11n mac frame aggregation mechanisms for next-generation high-throughput wlans. *IEEE Wireless Communications*, 15(1):40–47, February 2008.

[13] B. S. Kim, H. Y. Hwang, and D. K. Sung. Effect of frame aggregation on the throughput performance of ieee 802.11n. In *2008 IEEE Wireless Communications and Networking Conference*, pages 1740–1744, March 2008.

[14] Y. Lin and V. W. S. Wong. Wsn01-1: Frame aggregation and optimal frame size adaptation for ieee 802.11n wlans. In *IEEE Globecom 2006*, pages 1–6, Nov 2006.

[15] Tamizh Selvam and Subramanian Srikanth. A frame aggregation scheduler for ieee 802.11 n. In *2010 National Conference On Communications (NCC)*, pages 1–5. IEEE, 2010.

[16] Melody Moh, Teng-Sheng Moh, and Ken Chan. Error-sensitive adaptive frame aggregation in 802.11 n wlan. In *International Conference on Wired/Wireless Internet Communications*, pages 64–76. Springer, 2010.

[17] Anwar Saif, Mohamed Othman, Shamala Subramaniam, and Nor Asila Wati Abdul Hamid. An enhanced a-msdu frame aggregation scheme for 802.11 n wireless networks. *Wireless Personal Communications*, 66(4):683–706, 2012.

[18] Eitan Altman and Tania Jiménez. Novel delayed ack techniques for improving tcp performance in multihop wireless networks. In *IFIP International Conference on Personal Wireless Communications*, pages 237–250. Springer, 2003.

[19] Ajay Kumar Singh and Kishore Kankipati. Tcp-ada: Tcp with adaptive delayed acknowledgement for mobile ad hoc networks. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, volume 3, pages 1685–1690. IEEE, 2004.

[20] Ruy De Oliveira and Torsten Braun. A dynamic adaptive acknowledgment strategy for tcp over multihop wireless networks. In *INFOCOM 2005. 24th annual joint conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1863–1874. IEEE, 2005.

[21] Ruy De Oliveira and Torsten Braun. A smart tcp acknowledgment approach for multihop wireless networks. *IEEE Transactions on Mobile Computing*, 6(2):192–205, 2007.

[22] George Xylomenos, George C Polyzos, Petri Mahonen, and Mika Saaranen. Tcp performance issues over wireless links. *IEEE communications magazine*, 39(4):52–58, 2001.

[23] Michael Scharf, Marc Necker, and Bernd Gloss. The sensitivity of tcp to sudden delay variations in mobile networks. In *International Conference on Research in Networking*, pages 76–87. Springer, 2004.

[24] Andrei Gurtov. Effect of delays on tcp performance. In *Emerging Personal Wireless Communications*, pages 87–105. Springer, 2002.

[25] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A first look at quic in the wild. In *International Conference on Passive and Active Network Measurement*, pages 255–268. Springer, 2018.

[26] Jawad Manzoor, Idilio Drago, and Ramin Sadre. The curious case of parallel connections in http/2. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 174–180. IEEE, 2016.

[27] Jawad Manzoor, Idilio Drago, and Ramin Sadre. How http/2 is changing web traffic and how to detect it. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9. IEEE, 2017.

[28] J. Manzoor, R. Sadre, I. Drago, and L. Cerdà-Alabern. Is there a case for parallel connections with modern web protocols? In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9, May 2018.

[29] Linux Network Emulator. `http://man7.org/linux/man-pages/man8/tc-netem.8.html`, 2011. (accessed May 2019).

[30] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's quic protocol. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1193–1204. ACM, 2014.

[31] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is quic? provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231. IEEE, 2015.

[32] Quentin De Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 160–166. ACM, 2017.

[33] Alexander Rabitsch, Per Hurtig, and Anna Brunström. A stream-aware multipath quic scheduler for heterogeneous paths. In *ACM CoNEXT 2018 Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ'18)*, 2018.

[34] Sants-UPC Community Newtork. `http://sants.guifi.net`. (accessed May 2019).

[35] Community Networks Testbed for the Future Internet, CONFINE. `http://confine-project.eu/`. FP7 European Project 288535 (accessed May 2019).

[36] OpenWrt Linux distro. for embedded devices. `https://openwrt.org`. (accessed May 2019).

[37] Quick Mesh Project. `http://qmp.cat`. (accessed May 2019).

[38] Llorenç Cerdà-Alabern, Axel Neumann, and Leonardo Maccari. Experimental evaluation of bmx6 routing metrics in a 802.11 an wireless-community mesh network. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 770–775. IEEE, 2015.

[39] Open, Free and Neutral Network Internet for everybody. `http://guifi.net/en`. (accessed May 2019).

[40] Llorenç Cerdà-Alabern, Axel Neumann, and Pau Escrich. Experimental evaluation of a wireless community mesh network. In *The 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM'13*, Barcelona, Spain, November 3–8, 2013. ACM.

[41] qMp Sants-UPC monitoring page. `http://dsg.ac.upc.edu/qmpsu`. (accessed May 2019).

[42] Riccardo Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE Journal on Selected Areas in Communications*, 9(2):203–211, 1991.

[43] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[44] Florian Gratzer. Quic-quick udp internet connections. *Future Internet and Innovative Internet Technologies and Mobile Communications*, 2016.

Appendices

Appendix A QUIC features

- Multiplexing
  Stream multiplexing was implemented in HTTP/2 over TCP to solve the head-of-line blocking at the application layer: Even if HTTP/1.1 supports request pipelining, responses must arrive in the same order as the requests. Page rendering thus may get blocked if important requests are fired only after less important content is requested. However, HTTP/2 has only partially solved the problem and the head-of-line blocking has been passed down to the transport layer. When a packet is lost, all HTTP/2 streams are blocked and TCP buffers any subsequent packets until the successful retransmission of the lost packet. QUIC implements stream multiplexing at the transport layer. QUIC design avoids head-of-line blocking at both application and transport layer. A QUIC connection can still make progress on some streams while others are paused due to packet loss.

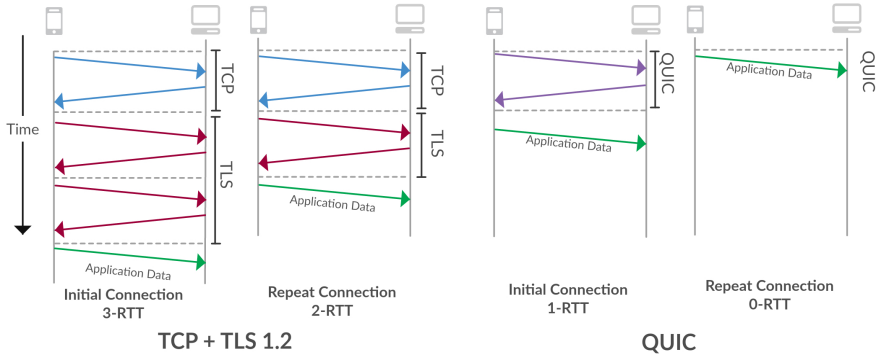


Figure 14: Comparison of TCP and QUIC connection establishment.

- Low-latency connection establishment
  The establishment of a secure connection usually requires several round trips. Round trip latency can make a big difference on long distance links or WiFi and cellular networks. QUIC combines the transport and crypto handshake and reduces the number of round trips required for setting up a connection.

  Fig. 14 shows the comparison of connection establishment of TCP and QUIC with encryption. It takes 3 RTT for TCP and TLS 1.2 to establish the connection with an unknown server. The first round trip is required for the three-way TCP handshake. The second round trip is required to negotiate the TLS version and the cipher suite. In the third round trip key exchange is initiated, which is used to establish the symmetric key for the session. In case of resumed connections, only 2-RTTs are required. This procedure is improved in TCP with TLS 1.3 where it takes 2-RTT for the initial connection and 1-RTT for resumed connections. QUIC further reduces this latency. It takes only 1-RTT to establish a secure connection with an unknown server using inchoate ClientHello (CHLO). It then starts application data transfer in the next round trip along with complete CHLO. Repeated connections are started with 0-RTT by sending the complete CHLO along with encrypted data directly.

- Connection migration
  QUIC supports connection migration e.g., from WiFi to cellular because QUIC connections are identified by a 64-bit connection ID which remains the same across these migrations. On the other hand, a TCP connection is identified by a 4-tuple (source and destination IP address, source and destination port number). Therefore, TCP connection does not survive IP address changes and NAT re-bindings.

- Header and payload encryption
  QUIC hides most of its state information by using header and payload encryption by default. While it helps in avoiding network ossification and pervasive monitoring, it comes at the cost of complicating network operations and management. Routine tasks of network operators such as detecting anomalies, capacity planning, and traffic engineering become harder due to transport layer header encryption.