QUIC Throughput and Fairness over Dual Connectivity*

David Hasselquist, Christoffer Lindström, Nikita Korzhitskii, Niklas Carlsson, Andrei Gurtov

Linköping University, Sweden

Abstract

Dual Connectivity (DC) is an important lower-layer feature accelerating the transition from 4G to 5G that also is expected to play an important role in standalone 5G radio networks. However, even though the packet reordering introduced by DC can significantly impact the performance of upper-layer protocols, no prior work has studied the impact of DC on QUIC. In this paper, we present the first such performance study. Using a series of throughput and fairness experiments, we show how QUIC is affected by different DC parameters, network conditions, and whether the DC implementation aims to improve throughput or reliability. Results for two QUIC implementations (aioquic, ngtcp2) and two congestion control algorithms (NewReno, CUBIC) are presented under both static and highly time-varying network conditions. Our findings provide network operators with insights and understanding into the impacts of splitting QUIC traffic in a DC environment. With reasonably selected DC parameters and increased UDP receive buffers, QUIC over DC performs similarly to TCP over DC and achieves optimal fairness under symmetric link conditions when DC is not used for packet duplication. The insights can help network operators provide modern users with better end-to-end service when deploying DC.

Keywords: QUIC, Dual Connectivity, Throughput, Fairness, Transport Protocol, Multipath

1. Introduction

The end-to-end performance depends on the interactions between protocols in different network layers. As new features are introduced on the lower layers, it is therefore important to understand the impact that such features and their parameters have on the upper layer protocols [2]. One such feature is Dual Connectivity (DC). DC was introduced in 4G, gained popularity with the introduction of 5G, and currently plays an integral role in accelerating the generational transition from 4G to 5G [3].

With DC, users can transmit and receive data from two base stations concurrently. This allows users to use both 4G and 5G radio networks in parallel, simplifying the above-mentioned generational transition. However, it has also been argued that DC should be a part of future 5G solutions needed to meet the requirements of Ultra-reliable and Low-Latency Communications (URLLC) [4, 5]. Combined with its increased usage, this has made DC an important 5G feature.

Like multipath transport protocols [6, 7, 8], DC can be used to combine WiFi with 4G and 5G solutions. Furthermore, like these protocols, DC can be used to achieve improved throughput (by sending different data over different paths), to increase reliability (by transmitting the same data over the different paths), or both. However, in contrast to the transport-layer multipath solutions, DC is performed within the link layer of the network stack and is therefore in practice invisible to transport layer protocols such as TCP and QUIC. This is an important observation since DC may introduce jitter or reordering of packets that can significantly impact TCP and QUIC performance.

In parallel with the transitioning of different cellular network generations, Google recently introduced QUIC as a next generation transport-layer solution aimed at addressing some shortcomings with TCP [9]. In contrast to TCP, QUIC is implemented in the user-space, on top of UDP, and provides much improved stream multiplexing compared to TCP. This is important to speed up web connections in the presence of packet losses and/or modern HTTP/2 traffic. Initial research shows that QUIC allows performance improvements over TCP in several cases while providing an easy way to achieve fast incremental deployment [9]. Popular services that already today use QUIC include Google search services, Chrome, Chromium, YouTube, and Facebook [9, 10].

Due to the increasing use and popularity of both QUIC and DC, combined with the continuous rollout of 5G networks using DC, it is important to understand how QUIC performs over DC under different network conditions, and the impact that different DC parameters have on QUIC performance.

In this paper, we present the first performance evaluation of QUIC over DC. First, a testbed is set up to simulate DC. The testbed captures QUIC and TCP performance under a wide range of network behaviors (based on bandwidth, delay, and loss conditions) and the impact of different DC parameters. Second, using a series of throughput and fairness experiments, we show how QUIC is affected by different DC parameters, network conditions, and whether the DC implementation aims to improve throughput or reliability. For our throughput eval-

*A shorter preliminary version of this work appeared in the IEEE MASCOTS 2020 workshop [1].

uation, we primarily compare the throughput of QUIC over DC with that of TCP over DC, and for our fairness comparisons we compare the throughput (and calculate a fairness index) of competing flows when using QUIC over DC. We also present results using different QUIC implementations (aioquic, ngtcp2), congestion control algorithms (NewReno, CUBIC), and for networks with both static and highly time-varying bandwidths. Our findings provide insights into the impact that DC and its parameters have on QUIC performance. Furthermore, we show the value of increasing the UDP receive buffers when running QUIC over DC, that QUIC over DC can achieve similar throughput as TCP over DC, and that QUIC over DC can achieve optimal fairness under symmetric link conditions, except if DC duplicates packets to increase reliability.

Overall, the experimental findings presented in this paper provide an important step towards understanding the interplay between DC and QUIC under different conditions and scenarios. Again, while much prior work have studied either DC or QUIC, we are the first to consider these in combination.

The remainder of the paper is organized as follows. Sections 2 and 3 introduce DC and present related works, respectively. The following sections present our methodology (Section 4) and performance evaluation results (Section 5), including a summary of our key findings, before we present our conclusions (Section 6).

2. Dual Connectivity

DC, also sometimes called inter-node radio resource aggregation, is a multi-connectivity technique introduced in release 12 of the third-generation partnership project (3GPP) [11]. The aim was to increase reliability, performance, and signaling due to frequent handovers in scenarios where macro and micro cells are connected with a non-ideal backhaul (X2) link. DC tries to achieve this by splitting the traffic over multiple paths.

Figure 1 shows an overview of DC in a Radio Access Network (RAN) environment. With DC, a User Equipment (UE) connects to two different network nodes, also known as Evolved Node Bs (eNBs) [5]. One of the network nodes will serve as Master eNB (MeNB), and the other one will serve as Secondary eNB (SeNB). Each of the MeNB and SeNB contains a separate Radio Link Control (RLC) and Media Access Control (MAC) layer, while sharing the same Packet Data Convergence Protocol (PDCP) layer.

DC is similar to carrier aggregation [12], but is performed in the PDCP layer instead of the MAC layer. Carrier aggregation uses the same scheduler for the separate connections and requires an ideal X2 link. The split connections are therefore often transmitted from the same node. In contrast, DC uses two separate packet schedulers together with a non-ideal X2 link, and packets are often originating from two different nodes.

PDCP is a sublayer located inside the link layer, just below the network layer and above RLC and MAC. The main tasks of PDCP are header compression and decompression, ciphering, integrity protection, transfer of data, sequence numbering, reordering and in-order delivery [13]. The PDCP layer can be

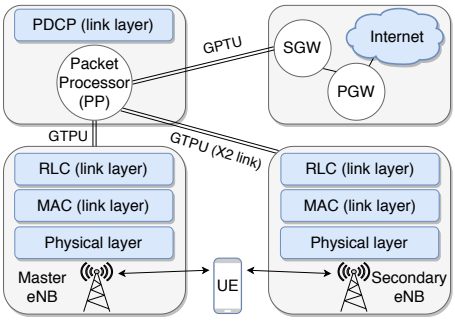


Figure 1: Dual connectivity overview.

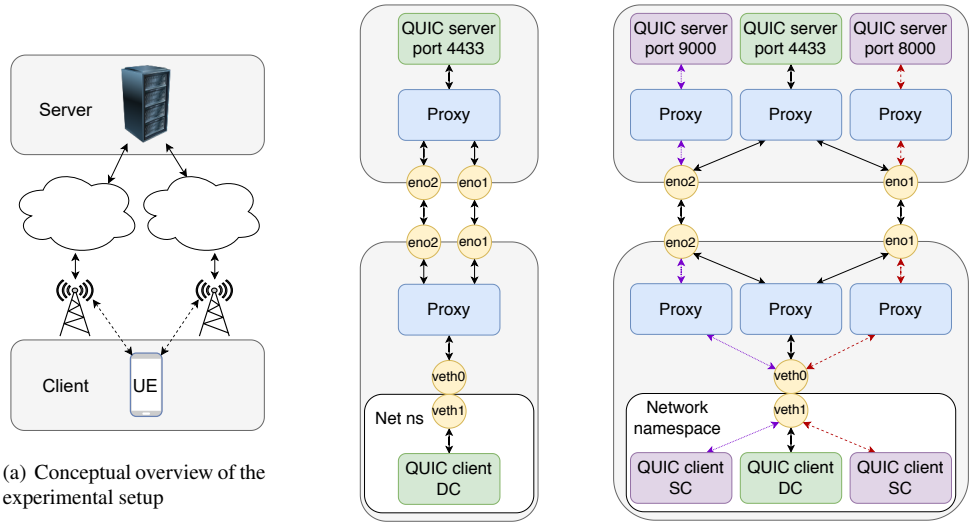
broken out into a unit called a Packet Processor (PP), which connects to Serving Gateway (SGW), MeNB and SeNB using a GTPU-tunnel. SGW is connected to the Packet Data Network Gateway (PGW), which in turn connects to the public internet. The PP can also be a part of MeNB. In this case, MeNB splits the traffic and the link between MeNB and SeNB becomes the X2 link. In both scenarios, the traffic is split in the PDCP layer.

3. Related Work

Dual connectivity: Unlike TCP, QUIC is relatively new, and there are few studies of it in specific scenarios such as DC. As QUIC shares similarities with TCP, we can obtain initial insights from research about DC that uses TCP as the transport protocol. Polese et al. [14] study the performance of TCP when using DC to perform mobile handovers for a UE and compare the performance with different single connection mobility management solutions. They show that DC can improve TCP goodput by quickly moving the traffic from one of the two DC links to the other.

Other studies have focused on specializations of DC; e.g., LTE-WLAN Aggregation (LWA) [15, 16], which allows for network traffic over LTE and WLAN. Jin et al. [15] propose LTE-W and show that splitting TCP over LTE and WiFi links at the PDCP layer can achieve similar throughput and better fairness than MP-TCP; demonstrating the value of lower-layer traffic splitting. Khadraoui et al. [16] investigate the effect of PDCP reordering when using TCP in LWA over heterogeneous links. Their results show that PDCP reordering can have adverse effects on TCP throughput, and that in some cases it is better to use only one link. Others have presented performance optimizations for DC [17, 18, 19, 20] but have not considered QUIC over DC. For example, Wu et al. [17] present an optimized DC traffic scheme for offloading the uplink of mobile users. Considering various tradeoffs, they achieve significant performance benefits compared to when using a fixed bandwidth allocation or scheduling scheme. At a high level, our study differs as we focus on QUIC and experimentally study its performance when using DC. While some works have looked at TCP with DC, no prior work has studied the performance of QUIC over DC.

Upper-layer multipathing: Multipathing is similar to DC but performed higher up in the network stack. Most such solutions are implemented in the transport layer, e.g., SCTP [8] and MP-TCP [7], but some are implemented in the network



(a) Conceptual overview of the experimental setup

(b) Throughput

(c) Fairness

Figure 2: Testbed for throughput and fairness tests.

layer [21]. Here, we focus on QUIC-based solutions. De Coninck and Bonaventure [6] implement Multipath QUIC (MP-QUIC) based on quic-go and lessons learned from MP-TCP, and show that serving QUIC over multiple paths is beneficial. Mogensen et al. [22] expands MP-QUIC to Selective Redundant MP-QUIC (SR-MPQUIC). Their solution modifies the congestion control algorithm, the scheduler, and the stream framer. SR-MPQUIC reduces latencies and improves reliability for priority data at a small increase in bandwidth usage and latencies for background data. The results show the importance of proper packet scheduling and the value of packet duplication. While additional cross-layer communication would be required to benefit DC, QUIC (and HTTP/3) also includes some unique attributes to assist packet/flow scheduling [23].

Fairness: Fairness can be difficult to judge when there are multiple paths with different amount of resources. As such, we take a look at studies on multipathing fairness. Becke et al. [24] study the fairness of different congestion control algorithms in multipath scenarios, targeting TCP-friendliness and focusing on two fairness types: link-centric fairness (where each flow share the link capacity equally), and network-centric flow fairness (where fairness of flows is considered in the whole network with several paths). Raiciu et al. [25] study how MP-TCP can replace single connections and load balancing in data centers with different network topologies. For specific topologies, MP-TCP significantly improved fairness and provided throughput closer to optimal compared to single connectivity using random load balancing. To judge the fairness, they and many others [25, 7, 26] evaluate multipathing using Jain's fairness index (JFI) [27]. Similar to these works, we use JFI here.

4. Methodology

4.1. Dual connectivity testbed

To evaluate the QUIC performance over DC, we used two machines in our evaluation framework: one machine to capture client-side behavior and performance, and one machine to capture server- and network-side effects. The two machines were connected via two network interface pairs (eno1 and eno2), each supporting 10 Gbps full duplex. To simulate different network conditions, tc in Linux was used to add extra delay, jitter, loss, and bandwidth limitations for each of the two links.

Depending on the specific experiment, we ran one or three QUIC/TCP clients on the client side and one or three QUIC/TCP servers on the server side. Figure 2 shows an overview of the testbed, starting with a high-level conceptual overview (Figure 2(a)) and then two sub-figures (Figures 2(b) and 2(c)) that captures the specific designs used for our throughput and fairness experiments. Furthermore, to simulate the functionality of DC and PDCP, two proxies were implemented for each QUIC client-server pair: one on the client side and one on the server side. Finally, each QUIC client was launched inside a network namespace (kernel functionality in Linux). This is to avoid the QUIC clients binding to a random interface and to ensure that we can control which interface each DC connection eventually is sent over. Here, two virtual interfaces were created to forward data to and from the namespace. We also note that there is no need for a network namespace on the server side, as the server will respond over the incoming interface, and DC is only studied on the downlink (server to client).

Table 1 specifies the hardware used in our experiments. We next describe the specific configurations used for our throughput and fairness tests, respectively, as well as our proxy implementation used to simulate DC and PDCP.

Throughput test configuration: For our throughput tests (Figure 2(b)), we used one client, one server, and studied the performance impact of DC parameters and the network conditions between them. In our baseline tests, both the QUIC server and QUIC client used aioquic [28]. When comparing with TCP, we used a Hypercorn server using HTTP/2 over TLS 1.3, and the client used curl to make HTTP/2 requests. As baseline, both TCP and QUIC used NewReno for congestion control.

Table 1: Hardware and operating systems used in tests.

Component	Client	Server
OS	Ubuntu 18.04.3 LTS	Ubuntu 18.04.3 LTS
Kernel	Linux 4.15.0-74-lowlatency	Linux 5.3.0-26-generic
Processor 1 & 2	Intel(R) Xeon(R) CPU E5-2690 v3	Intel(R) Xeon(R) CPU E5-2667 v3
eno1 & eno2	82599ES 10-Gigabit SFI/SFP+	82599ES 10-Gigabit SFI/SFP+

Fairness test configuration: For our fairness tests (Figure 2(c)), we used three clients and three servers. One end-to-end connection was performing DC, while the other two used single connectivity (SC) over interface eno1 and eno2, respectively. The server with port 8000 was operating only on eno1, while the server on port 9000 only on eno2. The QUIC server on port 4433 used DC and operated on both interfaces.

Proxy-based implementation: Since traffic splitting with DC is implemented in the link layer, QUIC (and TCP) are unaware that the traffic is sent over multiple paths, and therefore do not need to be modified. However, as DC was introduced for radio technology, the link layer functions and structure differ from the Ethernet links used here. To simulate the functionality of DC and PDCP, two proxies were implemented: one at the client and one at the server. First, to capture the PDCP functionality, packets originating from the server are caught by iptables OUTPUT chain and delivered to a NFQUEUE, before being read by the server proxy. The server proxy then adds a 2-byte PDCP sequence number to each packet and routes the packets to the client over two interfaces. When running DC, the server proxy alternates between the two interfaces.

At the client proxy, packets are caught in the PREROUTING chain and delivered to NFQUEUE. The client proxy can then read from the queue, perform PDCP convergence of the two streams, do PDCP reordering, and remove the sequence numbers that were added by the server proxy.

If a packet is received in order, it is immediately forwarded to the client. However, if a packet is out of order, it is kept until the missing packets are processed or until a PDCP timer of 200 ms is reached. If the timer is reached, all packets before the missing packet and all consecutive packets after the missing packets are delivered. The reordering algorithm follows the PDCP standard described in 3GPP [13] and the testbed was developed in close consultation with Ericsson. Our proxy adds around 1 ms to the total RTT, assuming that the packets arrive in order. Without PDCP, large reordering occurs, resulting in QUIC having a very low throughput.

4.2. Performance testing

To understand how DC affects QUIC, a series of tests are performed that captures the impact of different DC parameters and network conditions. In our experiments, we vary one parameter at a time, starting with a default configuration, while keeping the others constant (as per the default configuration). In the throughput tests, the client downloads a 100MB file, and in the fairness tests each client downloads a 1GB file and we measure the clients' performance for the first three minutes of the download. For each test configuration, we run ten tests and calculate both average and standard deviation values for the metrics of interest (i.e., throughput and JFI). When discussing JFI, we also stress that there is no global JFI threshold that can be considered as a threshold for what is "fair" and "not fair". Rather than suggesting any threshold, we instead focus on relative comparisons as a parameter changes or between the JFI of two alternative configurations operating under otherwise similar conditions.

DC parameters and default configurations: The primary DC parameters we varied were the DC batch size and DC batch split. These parameters determine how many packets are sent over each interface before the server proxy switches to the other interface. For example, with a DC batch size of 100 and a DC ratio of 9:1 (90% eno1 and 10% eno2), the proxy would send 90 packets over eno1, before switching over to send 10 packets over eno2. In our default experiments, the default DC batch size and DC ratio was configured to 100 and 1:1, respectively.

Network emulation parameters and default configurations: To capture different network conditions, we primarily varied the bandwidths, delays, and loss rates of the links. For both the bandwidths and delays, we present experiments both where we vary the average values and where we vary the ratio between the two links. In the case we vary one of the ratios, we keep the average value of that metric constant. For example, a bandwidth ratio of 3:1 corresponds to 30 Mbps and 10 Mbps for the downlink interfaces eno1 and eno2, respectively. In our default experiments, each link operates at 20 Mbps and has normally distributed per-packet delays with a mean of 10 ms and a coefficient of variation of 10%.

QUIC and TCP configurations/versions: Throughput tests for QUIC are performed both with the default UDP receive socket buffer size and a larger receive buffer size. The larger size is used to give a fair comparison to TCP, as the kernel performs buffer autotuning for TCP [29]. When studying fairness, QUIC with modified buffer size is used and the fairness is calculated using JFI. Furthermore, we use Qvis and Qlog [30, 31] to debug and study QUIC at a more detailed level.

In our default scenarios we use aioquic with NewReno. However, as discussed by McMillan and Zuck [32], the QUIC RFC is ambiguous, open for interpretation, and differences between QUIC implementations following the same RFC have been demonstrated using specification testing. Marx et al. [33] also highlight this diversity by comparing 15 different QUIC implementations and showing that they are highly heterogeneous. We therefore repeated our experiments with both another QUIC implementation (ngtcp2 [34]) and congestion control algorithm (CUBIC).

Trace-based evaluation: Finally, to capture a use case with varying bandwidth, experiments were repeated using a real

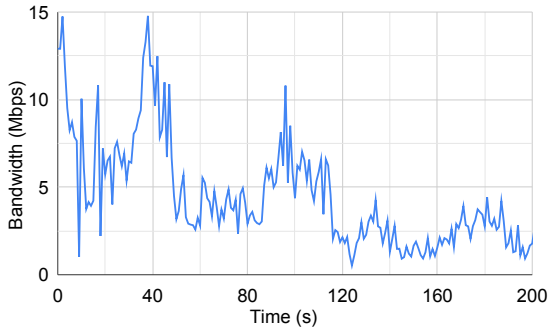


Figure 3: Example LTE bandwidth trace [35]

Table 2: Average throughput and RTT with and without PDCP proxy. Standard deviations within parentheses.

	Without proxy	**With proxy**
RTT (ms)	56.01 (0.62)	58.07 (0.85)
Throughput (Mbps)	18.28 (0.17)	17.00 (0.30)

LTE bandwidth trace collected by Raca et al. [35]. The specific bandwidth trace (*Static A_2018.02.12_16.14.02*) is used for both downlinks and has an average throughput of 4.5 Mbps for the first 200 seconds. Figure 3 shows the sampled bandwidth trace used as a function of time.

4.3. Dual connectivity overhead

To evaluate the overhead invoked by our testbed simulating DC, we perform 10 independent measurements using our default configuration and capture both the bandwidth and RTT with and without using our PDCP proxies. When running without a proxy, packets are being passed directly to the interfaces, skipping the NFQUEUE. Table 2 shows the evaluation results with their standard deviations. When using PDCP proxy, the average RTT increases with 2 ms. This corresponds to 1 ms per proxy, as packets are passed through both a client and server proxy. The increase of RTT causes the throughput to drop from 18.28 Mbps to 17.00 Mbps. This shows that only small performance overheads are added from our DC simulations.

5. Evaluation Results

In this section, we present our performance evaluation results under different scenarios and configuration. First, we show the impact of DC parameters that network operators control (Section 5.1). Then, we present our experiments under different network conditions (Section 5.2), before showing the impact of using DC to duplicate packets and improve reliability (Section 5.3). Next, we repeat our experiments using another QUIC implementation and congestion control algorithm (Section 5.4), and over a variable bandwidth scenario (Section 5.5). Finally, we give a summary of the main observations from our results (Section 5.6). Table 3 summarizes our experiments and provides an overview of the presented figures in this section.

5.1. Dual connectivity parameters

DC batch size: When using DC, network operators must select a good DC batch size for each connection. To illustrate the impact of this choice on QUIC performance, Figure 4(a) shows the throughput as a function of the DC batch size. (We omitted the standard deviations from all figures, as they are small; e.g., well within 1 Mbps in more than 90% of the cases.) In general, large DC batch sizes result in lower throughput. One reason for this is reduced link utilization. For example, Figures 4(b) and 4(c) show the link utilization of the two links when using a DC batch size of 50 and 500, respectively. With a large DC batch size, we see significant periods during which one of the links is underutilized as almost all packets are being forwarded over the other interface. With smaller DC batch sizes, both links can better be used concurrently. However, there is also a penalty to using too small batch sizes, as this increases the number of re-order events. The best batch sizes are instead typically in the mid-range (e.g., around 100-150), with the sweet spot depending on the protocol being used. Finally, we note that QUIC with modified buffers performs similar to TCP for much of the parameter range.

Figure 5(a) shows summary results for our fairness tests with varying DC batch sizes. Here, we measure fairness using Jain's fairness index (JFI), shown using purple text, as averaged over 10 full runs. When discussing fairness, it is important to note that the relative throughput of the competing clients can vary significantly over time. This is illustrated in Figure 5(b) where we show example throughput for the three competing clients over a 3-minute long experiment with the default settings. Here, it is important to note that the 3-minute time frame (duration of x-axis) is 60 times greater than the total duration shown for the example traces shown in Figures 4(b) and 4(c). We observe that short-term variations in the fairness persist up to time scales of about 30 seconds. By using 3-minute traces and reporting averages over 10 runs, such temporal unfairness are filtered out. As an additional reference point, Appendix A presents example traces with pairs of competing SC clients: (i) QUIC vs. QUIC, (ii) QUIC vs. TCP, and (iii) TCP vs. TCP.

Similar to the throughput, the fairness is negatively affected by large DC batch sizes. In fact, the user using DC observe a significant throughput reduction with batch sizes of 150 and above. Here, SC clients can monopolize the links during the DC client's off periods, while DC is always sharing the link it is currently sending to at every point of time. This allows SC clients to increase their congestion window further than the DC client and to use a larger bandwidth share.

DC batch split: Network operators also control the DC ratio. This parameter determines the split over the two links. In contrast to Figures 4(b) and 4(c), Figure 6 illustrates an unbalanced example with a DC batch split of 9:1, in which 90% of the packets are sent over the main interface (eno1), and Figure 7(a) shows the throughput as a function of the percent of packets sent over eno1. As per our default case, both links have the same network conditions. Compared with the utilization examples of balanced DC ratio (Figures 4(b) and 4(c)), we note that the less loaded link almost never reaches its full capacity; not even during burst periods. This results in sub-optimal link utilization in which only one of the links are well-utilized (close to 100%). This case corresponds to the 10% case (and by symmetry the 90% case) shown in Figure 7(a). More generally, the

Table 3: Performance evaluation results overview.

Class	Parameter	QUIC impl.	CC algorithm	Throughput Fig.	Fairness Fig.
DC parameters (Section 5.1)	DC batch size	aioquic	NewReno	4(a)	5(a)
	DC batch split			7(a)	8(a)
Network conditions (Section 5.2)	BW ratio			7(b)	8(b)
	BW & DC ratio			7(c)	8(c)
	Low delay ratio			9(a)	10(a)
	High delay ratio			9(b)	10(b)
	Loss rates			9(c)	10(c)
Duplicate packets (Section 5.3)	Loss rates			11(a)	11(b)
QUIC implementation and congestion control algorithm (Section 5.4)	DC batch size	ngtcp2	NewReno	12(a)	12(b)
			CUBIC		12(c)
	DC batch split		NewReno	13(a)	14(a)
			CUBIC		15(a)
	High delay ratio		NewReno	13(b)	14(b)
			CUBIC		15(b)
	Loss rates		NewReno	13(c)	14(c)
			CUBIC		15(c)
	BW ratio		NewReno	16(a)	16(b)
			CUBIC		16(c)
	BW & DC ratio		NewReno	17(a)	17(b)
			CUBIC		17(c)
	Low delay ratio		NewReno	18(a)	18(b)
			CUBIC		18(c)
	Loss rates with duplicate packets		NewReno	19(a)	19(b)
			CUBIC		19(c)
Bandwidth variability (Section 5.5)	DC batch size	aioquic	NewReno	20(a)	N/A*
		ngtcp2	NewReno + CUBIC	20(d)	
	High delay ratio	aioquic	NewReno	20(b)	
		ngtcp2	NewReno + CUBIC	20(e)	
	Loss rates	aioquic	NewReno	20(c)	
		ngtcp2	NewReno + CUBIC	20(f)	
	DC batch split	aioquic	NewReno	21(a)	
		ngtcp2	NewReno + CUBIC	22(a)	
	Low delay ratio	aioquic	NewReno	21(b)	
		ngtcp2	NewReno + CUBIC	22(b)	
	Loss rates with duplicate packets	aioquic	NewReno	21(c)	
		ngtcp2	NewReno + CUBIC	22(c)	

* After validating that the throughput results held with variable bandwidth conditions, it was decided to not run a full set of corresponding fairness tests.

throughput peaks when using a 50/50 split, and decreases as a convex function as the split becomes more uneven. The decrease in throughput caused by poor link utilization of the less loaded link (eno2 in Figure 6) highlights the value of careful batch split selection to best achieve the full potential of DC.

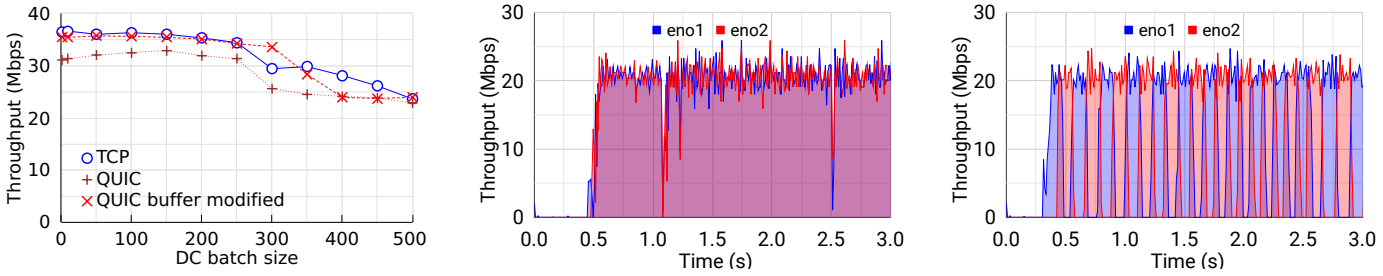
Figure 8(a) shows our corresponding fairness results. When the ratio is significantly skewed (e.g., below 20% or above 80%), the throughput of the SC with the higher throughput increase/decrease at roughly the same rate as the DC's throughput increase/decrease, whereas the other SC with lower throughput has fairly constant throughput over these skewed splits. In this region, the DC compete (almost) fairly only over the more utilized interface. As the DC split becomes more even, the overall fairness improves, with optimal fairness and all connections having roughly equal throughput when perfectly balanced.

5.2. Network conditions

We next look closer at the impact of the network conditions. Again, we consider one parameter at a time, keeping the other parameters fixed as in our default case.

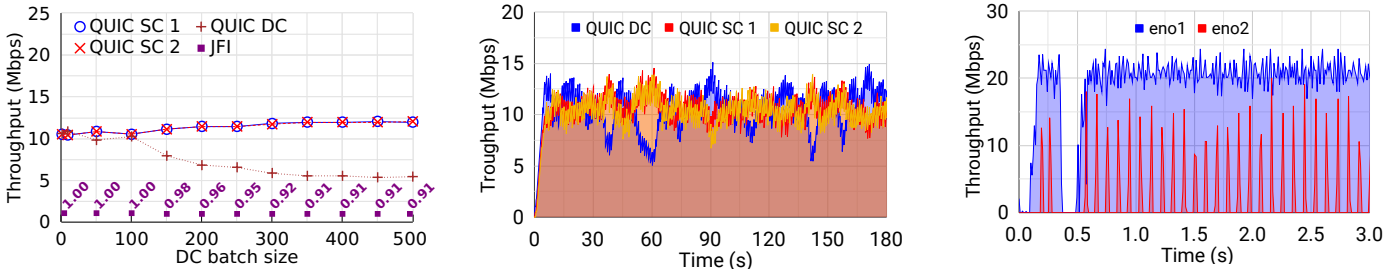
Bandwidth ratio: Figures 7(b) and 7(c) show the throughput for different bandwidth ratios. Figure 7(b) shows results for the case when the batch split is 50/50, and Figure 7(c) shows results for when the batch split is selected to match the bandwidth ratio. Figure 7(b) illustrates the importance of matching ratios, as the highest throughputs are achieved with a ratio of 1:1. As the ratio increases, a 50/50 batch split underutilizes the link with higher bandwidth. In contrast, when the DC batch split is selected to match the bandwidth ratio (Figure 7(c)), a much better overall throughput is achieved. With QUIC buffer modified and TCP, the impact is very small. The reason for the worse performance of QUIC with default buffers is the higher burstiness caused by increased reordering. Despite PDCP mitigating reordering, it results in increasing RTTs.

The fairness results for the cases when we vary the bandwidth ratio of the two links are shown in Figures 8(b) and 8(c). Similar to the throughput results, relatively higher fairness is



(a) Impact of DC batch size (b) Link utilization with batch size of 50 (c) Link utilization with batch size of 500

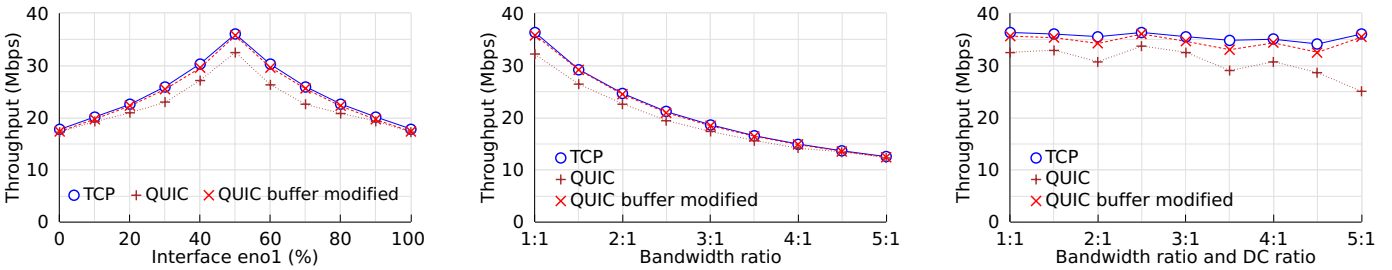
Figure 4: Throughput and link utilization for different DC batch sizes



(a) Impact of DC batch size (b) DC fairness example

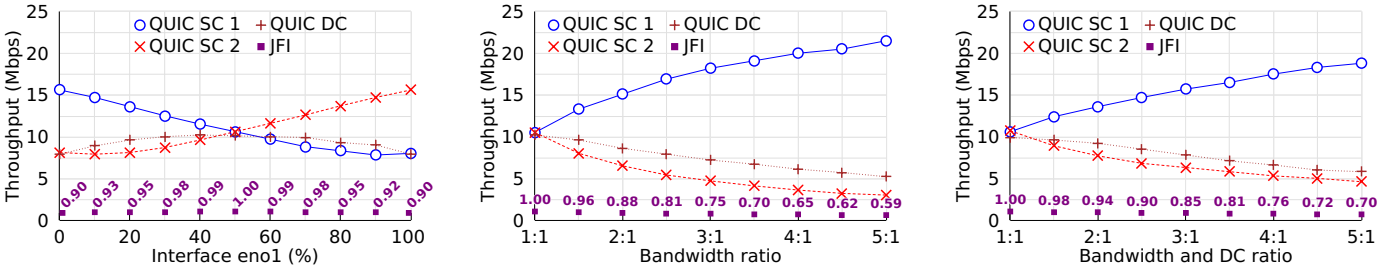
Figure 5: Fairness for different DC batch sizes

Figure 6: Link utilization example with unbalanced DC ratio of 9:1



(a) DC batch split (b) BW ratio (c) BW & DC ratio

Figure 7: Throughput for different batch splits and bandwidth ratios



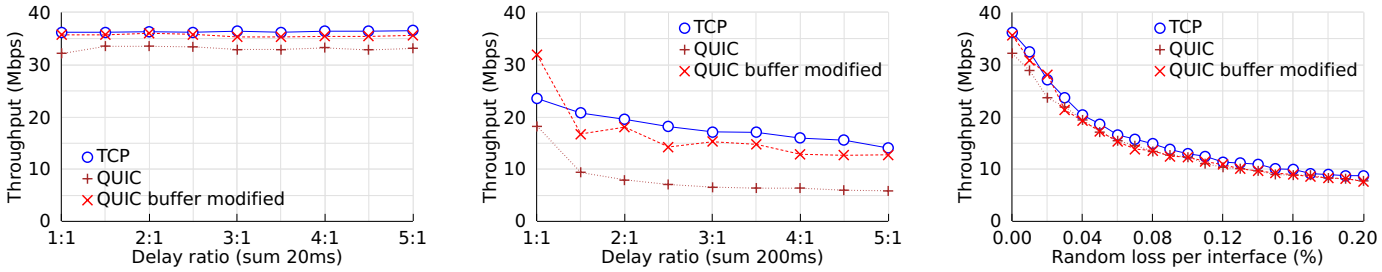
(a) DC batch split (b) BW ratio (c) BW & DC ratio

Figure 8: Fairness for different batch splits and bandwidth ratios

achieved when the DC split is selected based on the capacity of the two links. For example, even when the bandwidth ratio is 5:1, the scenario in which the DC split matches the bandwidth ratio achieves a JFI of 0.70, compared to 0.59 in the case a 50/50 split is used. In both cases, the bandwidth usage is dominated by the SC user with higher bandwidth and the DC user relies heavily on the throughput achieved via the weaker link. However, the fairness improves as DC moves more traffic to the link with the higher bandwidth. Yet, the highest fairness (JFI=1.00) is achieved only when the ratios are equal.

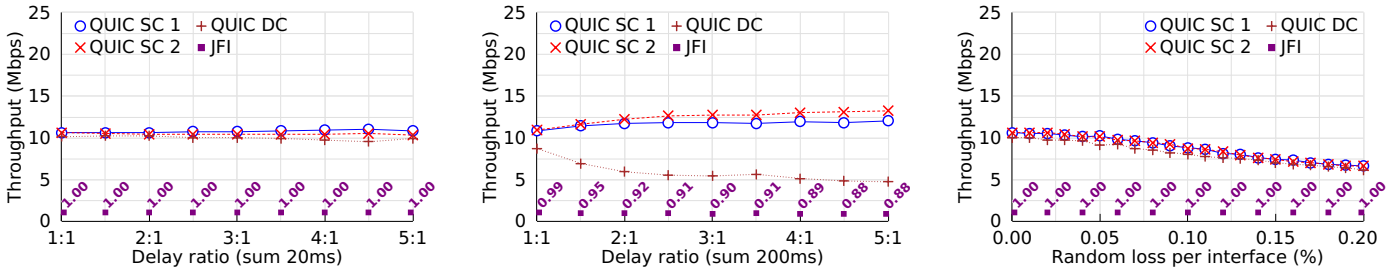
Delay ratio: Both the throughput and fairness are negatively affected by increasing delays, and in the case of a high average delay, these metrics are also negatively affected by an increasing delay ratio. This is illustrated by comparing the throughput Figures 9(a) and 9(b) or fairness Figures 10(a) and 10(b). For both types of experiments, the two figures show results for low-delay and high-delay scenarios, respectively. In the low-delay scenarios, the sum of the delays over the two links is 20 ms, and in the high-delay scenario the sum is 200 ms.

The throughput decrease is mostly due to increased packet reordering caused by the higher delays. In these cases, the PDCP layer will buffer more packets before performing a batch deliv-



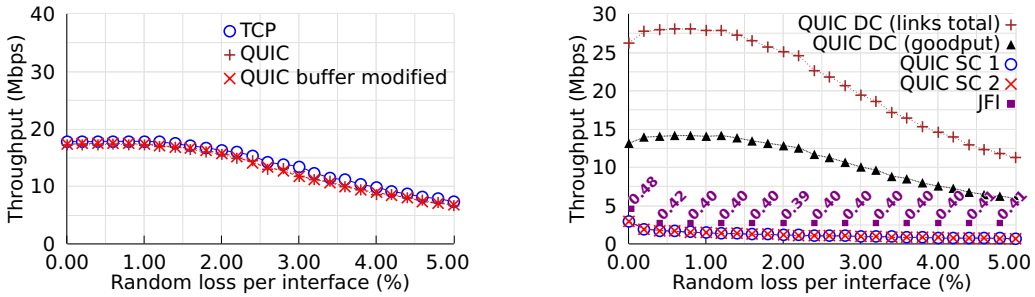
(a) Low delay ratio (b) High delay ratio (c) Random loss

Figure 9: Throughput for different delay and loss ratios



(a) Low delay ratio (b) High delay ratio (c) Random loss

Figure 10: Fairness for different delay and loss ratios



(a) Throughput (b) Goodput fairness

Figure 11: DC throughput and fairness with duplicate packets

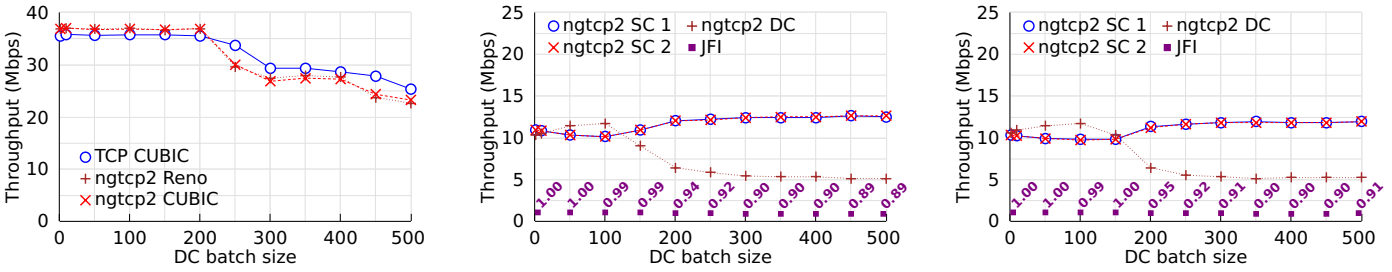
ery to the QUIC client, causing packet bursts as well as a higher RTT. Furthermore, after receiving a batch delivery, the clients will send a cumulative ACK for many packets, which will, for a short time, largely decrease the number of packets in flight when received at the server. The draft for QUIC [36] recommends a pacer, which helps the QUIC server recover from an ACK-burst by sending new packets at a steadier pace. The advantage of more even pacing can be seen by the higher values observed with a delay ratio of 1:1 in Figure 9(b).

The increasing delays and delay ratios also negatively impact fairness. For example, in the low-delay case (Figure 10(a)), JFI reduces from 0.9996 to 0.9986 (both rounded to 1.00 in the figure) as the delay ratio increases from 1:1 to 5:1, whereas JFI drops from 0.9903 to 0.8758 for the high-delay case (Figure 10(b)). While the reductions of the fairness index for the high-delay case are significant, these reductions are still much smaller than those observed when increasing the bandwidth rations equally (e.g., Figures 8(b) and 8(c)). These relative comparisons suggest that fairness is less affected by network-delay-ratio differences than by equally large bandwidth-ratio differences even under the large-delay case. The higher throughput of SC 2 compared to that of SC 1 is due to its lower RTT.

Loss rates: While increased packet losses negatively impact the throughput (Figure 9(c)), small packet losses have very limited impact on the fairness index (Figure 10(c)).

5.3. Use of duplicate packets

Thus far we have considered a simple use case in which DC primarily is used to improve throughput and no packet is sent over both interfaces. Besides improving throughput, DC can also be used to increase connection reliability. In Figure 11(a), we show the throughput when duplicating every packet and sending them on two separate paths. Compared to previous cases where we use DC to improve throughput and obtain an aggregated bandwidth of 40 Mbps, here we are only able to obtain a maximum bandwidth capacity of 20 Mbps, as the two links are used to send redundant data. This clearly shows the tradeoffs and importance of balancing the throughput and reliability. However, DC with packet duplication negatively affects fairness. For example, in fairness tests with loss rates of 0-to-5% (Figure 11(b)) JFI is in the range from 0.39 to 0.48. For DC in Figure 11(b), we show both the combined interface throughput (B) and the goodput (X), which under an independence model with retransmissions (after simplification) can be



(a) Throughput (b) Fairness with NewReno (c) Fairness with CUBIC

Figure 12: Impact of congestion control algorithm (NewReno and CUBIC): Performance examples with ngtcp2 as QUIC version for different batch sizes

related as $X = B(1+p)/2$, where p is the loss rate. The low fairness stems from DC having a much higher end-to-end packet delivery probability (i.e., $1-p^2$ vs. $1-p$ under independence assumptions) and lower end-to-end packet loss probability (i.e., p^2 vs. p) compared to SC. This results in DC having less end-to-end packet losses, and obtaining a much larger share of the link bandwidths. As SC packets are not duplicated and have no redundancy, packet losses will lead to much performance degradation. These results show that packet duplication can provide much higher reliability at the cost of fairness and goodput. We note that with 0% added random loss per interface, packet loss still occur due to full buffers dropping packets (e.g., from congestion control bandwidth probing). When duplicating packets, QUIC DC is able to better recover and obtain a large bandwidth share, while QUIC with SC struggles to recover, causing high unfairness.

5.4. QUIC implementation and congestion control algorithm

To explore the impact of other QUIC implementations and congestion control algorithms, experiments were repeated using (1) the QUIC implementation named ngtcp2, and (2) the congestion control algorithm CUBIC. For throughput experiments, Figure 12(a) shows little to no differences in the results between different congestion control algorithms when varying DC batch size. However, when compared to Figure 4(a), differences can be observed between the QUIC implementations. While the results follow the same patterns for both implementations, the considerable throughput drop occurs at different batch sizes. Another noticeable difference is that ngtcp2 has a slightly higher throughput than aioquic at smaller batch sizes, exceeding the throughput for TCP.

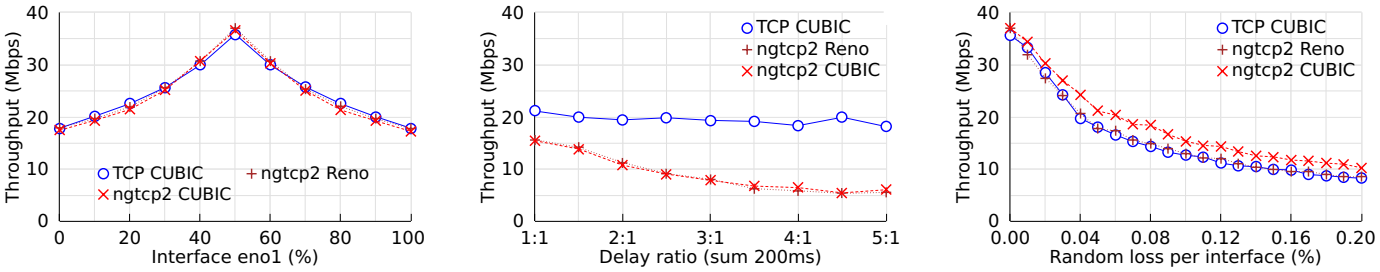
Figures 12(b) and 12(c) show the corresponding fairness results for different DC batch sizes. Again, only small differences between the NewReno and CUBIC results are observed. For example, the JFI differ by at most 0.02 (DC batch size of 500) between the algorithms. DC using CUBIC is initially slightly more resilient to performance drops occurring with a larger batch size. In contrast, NewReno allows the SC connections to achieve slightly higher throughput at larger batch sizes while the DC throughput is similar to CUBIC at higher batches. When comparing Figures 12(b) and 12(c) to 5(a), some differences can be seen between the QUIC implementations. Ngtcp2 is more aggressive, leading to the DC connection having slightly higher throughput than the SC connections at smaller batch sizes and a drastic reduction in throughput when the batch size increases. Aioquic has a more balanced sharing of the bandwidths at smaller batch sizes and see a smaller reduction in throughput at larger batch sizes.

When studying the DC batch split using ngtcp2 and different congestion control algorithms (Figure 13(a)), minimal difference in the overall throughput is observed. Compared to aioquic in Figure 7(a), little differences are observed at more uneven ratios. Ngtcp2 achieves a higher throughput than aioquic and TCP at more balanced ratios. Comparing to the corresponding fairness results in Figures 14(a) and 15(a) to 8(a), larger differences is seen between the QUIC implementations. While the two implementations exhibit similar behavior at the most uneven split, the DC connection using ngtcp2 grows more aggressively than the aioquic counterpart when the ratio becomes more balanced. This growth leads to optimal fairness for aioquic, but results in a slightly unfair bandwidth allocation for ngtcp2.

A significant throughput difference between the QUIC implementations can be seen when comparing the high delay ratio experiments in Figures 13(b) and 9(b). Ngtcp2 achieves a significantly lower throughput than aioquic throughout the experiment. This is most likely due to differences in the pacer implementations. Differences can also be seen when comparing corresponding fairness tests (Figures 14(b) and 15(b) vs. Figure 10(b)). Here, the DC connection using ngtcp2 achieves more fair throughput than the aioquic counterpart at balanced ratios and sees a slower drop when the ratio gets skewed. However, after the 2:1 ratio point, the DC connections' throughputs become the same for the two implementations. The ngtcp2 SC connection with a higher delay has a much worse performance than the aioquic counterpart at more skewed ratios.

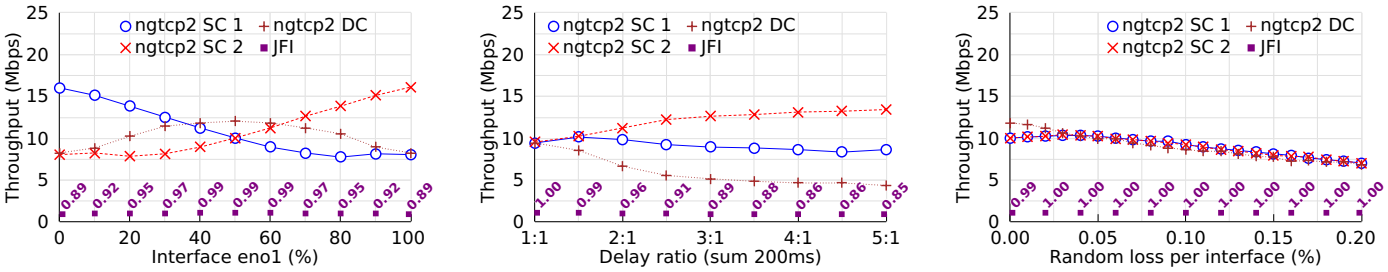
Finally, for loss rates using ngtcp2 and CUBIC, we observe only small differences compared to our default scenario with aioquic and NewReno (throughput Figure 13(c) compared to Figures 9(c), and fairness Figures 14(c) and 15(c) compared to Figure 10(c)). However, in contrast to the other experiments, ngtcp2 using CUBIC shows a noticeable better performance compared to TCP CUBIC. Looking closer at the 0.08% loss case in Figure 13(c), we observe that the TCP implementation more often stays in CUBIC's TCP mode (used when detecting growth slower than Reno counterparts, e.g., due to low bandwidth delay products). This also explains why TCP CUBIC, TCP NewReno, and ngtcp2 NewReno perform similarly here.

In general, ngtcp2 achieves higher throughput than aioquic, even though both follow the same IETF recommendations. As discussed, differences can occur due to the RFC being open for



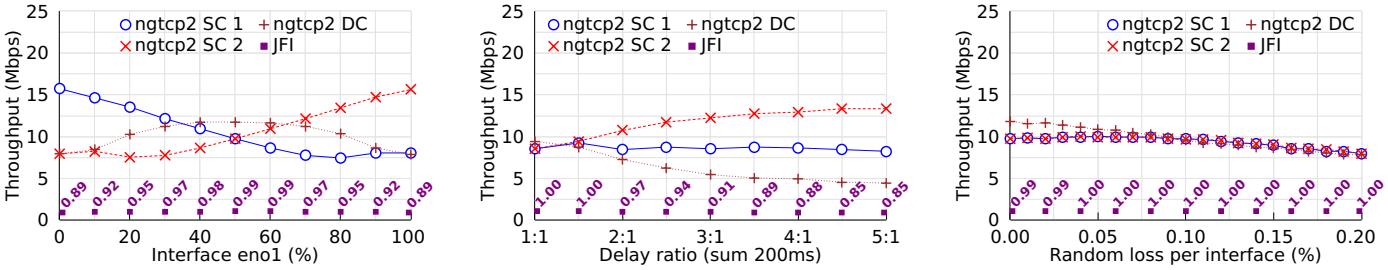
(a) DC batch split (b) High delay ratio (c) Random loss

Figure 13: Throughput when using ngtcp2 (as QUIC implementation) with NewReno and CUBIC (as congestion control algorithms) for different parameters



(a) DC batch split (b) High delay ratio (c) Random loss

Figure 14: Fairness when using ngtcp2 (as QUIC implementation) with NewReno (as congestion control algorithm) for different parameters



(a) DC batch split (b) High delay ratio (c) Random loss

Figure 15: Fairness when using ngtcp2 (as QUIC implementation) with CUBIC (as congestion control algorithm) for different parameters

interpretation. The execution speed and resources required by the two implementations also differ. Ngtcp2 is implemented in C and aioquic in Python. With ngtcp2, a larger receive buffer did not impact throughput, as the client buffer was quickly emptied. Ngtcp2 is also noted to be greedier than aioquic over DC, often introducing some unfairness to scenarios that were fair for aioquic. One potential reason is the difference in pacer implementation, as the IETF only recommends a pacer but does not specify it in detail. The difference in pacer implementation is also clearly shown in high delay ratio tests.

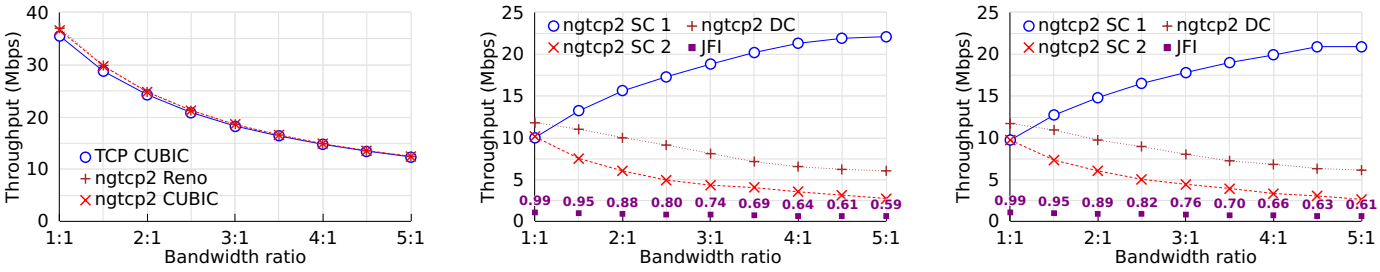
Additional tests using ngtcp2 with different congestion control algorithms: To demonstrate the generality of our observations, we next present additional throughput and fairness results when using ngtcp2 with NewReno and CUBIC.

Figure 16 shows the throughput and fairness impact of the bandwidth ratio when keeping the DC ratio fixed. (The corresponding sub-figures for aioquic with NewReno are Figures 7(b) and 8(b). Aioquic does not support CUBIC.) Figure 17 shows the impact of the bandwidth ratio when using a matching DC ratio. (The corresponding sub-figures for aioquic with NewReno are Figures 7(c) and 8(c).) Figure 18 shows the impact of the delay ratio for our low-latency scenarios. (The corresponding sub-figures for aioquic with NewReno are Figures 9(a) and 10(a).) Comparing with the corresponding results using aioquic, in all three cases, the observation from Section 5.4 holds true, including that ngtcp2 is more aggressive and achieves higher throughput than aioquic. The results are consistent regardless of whether NewReno or CUBIC is used as congestion control algorithm.

Finally, we note that the conclusions regarding the protection that duplicate packets offer during lossy scenarios are consistent across QUIC versions and congestion control algorithms. In all cases, duplication provides better loss protection at the cost of fairness and goodput. Also in this special scenario does ngtcp2 achieve higher throughput than aioquic. These results are shown in Figure 19 and can be compared to Figure 11.

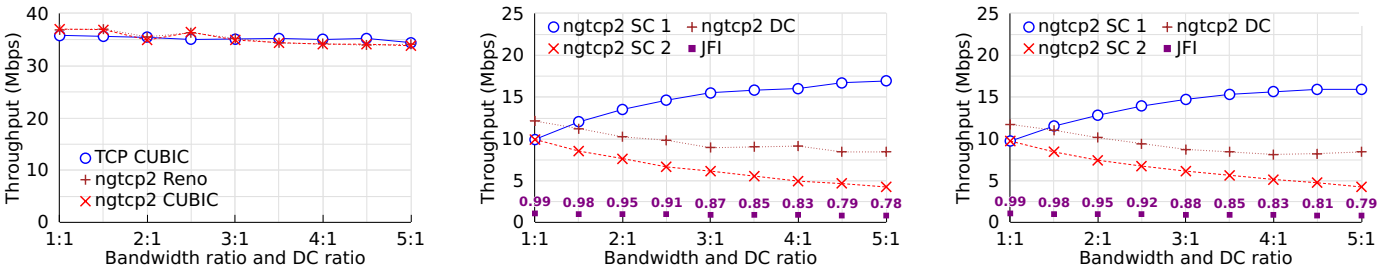
5.5. Bandwidth variability scenario

Baseline comparisons: To capture a more realistic bandwidth user scenario, Figures 20(a) to 20(c) show repeated experiments with aioquic for DC batch size, high delay ratio and loss rates performed over a LTE sampled bandwidth trace. Figures 20(d) to 20(f) show these results but using ngtcp2 with CUBIC. For DC batch size and loss rates, similar trends are observed as when using a fixed bandwidth capability. Similar trends are also observed in the case of delay ratio, but with



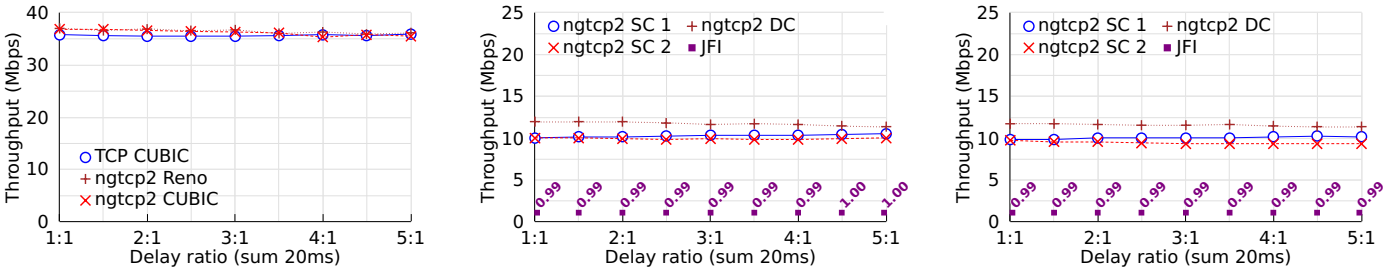
(a) Throughput (b) Fairness with NewReno (c) Fairness with CUBIC

Figure 16: Impact of the bandwidth ratio when using ngtcp2 as QUIC version with two different congestion control algorithms: NewReno and CUBIC



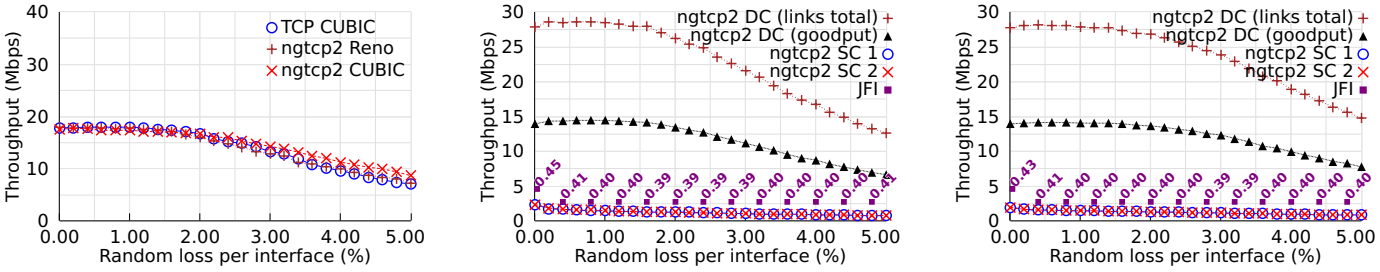
(a) Throughput (b) Fairness with NewReno (c) Fairness with CUBIC

Figure 17: Impact of the bandwidth ratio when having a matching DC ratio and using ngtcp2 as QUIC version with two different congestion control algorithms: NewReno and CUBIC



(a) Throughput (b) Fairness with NewReno (c) Fairness with CUBIC

Figure 18: Impact of the delay ratio in a low-delay scenario, when using ngtcp2 as QUIC version with two different congestion control algorithms: NewReno and CUBIC



(a) Throughput (b) Fairness with NewReno (c) Fairness with CUBIC

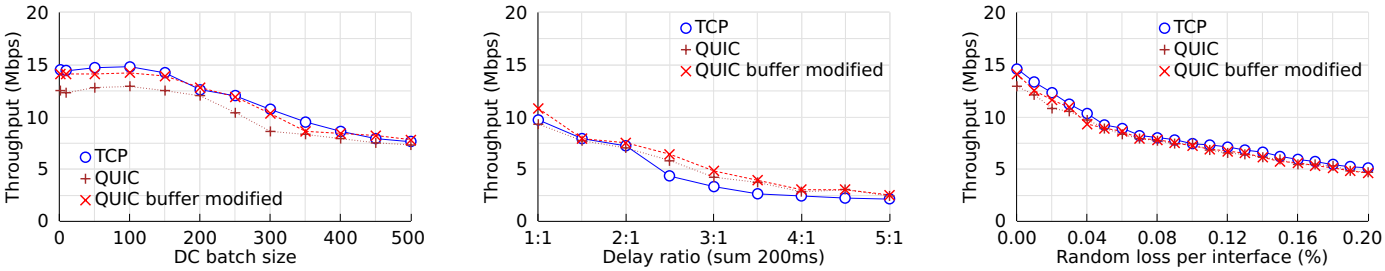
Figure 19: DC throughput and fairness when using duplicate packets and ngtcp2 as QUIC version with two different congestion control algorithms: NewReno and CUBIC

the effect of the pacer more clearly shown. Lastly, we note that ngtcp2 is more aggressive than aioquic and that CUBIC achieves higher throughput.

Additional trace-based results: Figures 21 and 22 show additional trace-based results using the two QUIC versions aioquic and ngtcp2, respectively. Here, we show results for (a) the impact of the DC ratio, (b) the impact of the delay ratio when operating in the low-latency range, and (c) the loss protection provided by the use of duplicate packets in loss scenarios. While the throughput differences here are smaller between ngtcp2 and aioquic than for the other scenarios (Figures 21 and 22 compared to Figure 20), as well as between CUBIC and NewReno (Figure 22), also for these cases do we observe the same trends as to when using a fixed bandwidth capability.

5.6. Summary of results

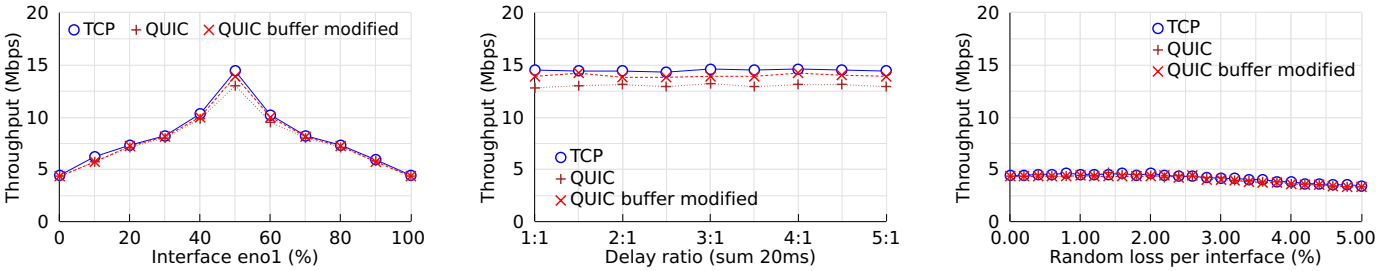
Throughput: In general, QUIC performs better with a larger receive buffer size. With a modified buffer, it reaches similar throughput as the TCP counterparts. For DC parameters batch size and batch split, the throughput is negatively affected by a



(a) DC batch size, aioquic (repeated Fig. 4(a)) (b) High delay ratio, aioquic (repeated Fig. 9(b)) (c) Random loss, aioquic (repeated Fig. 9(c))

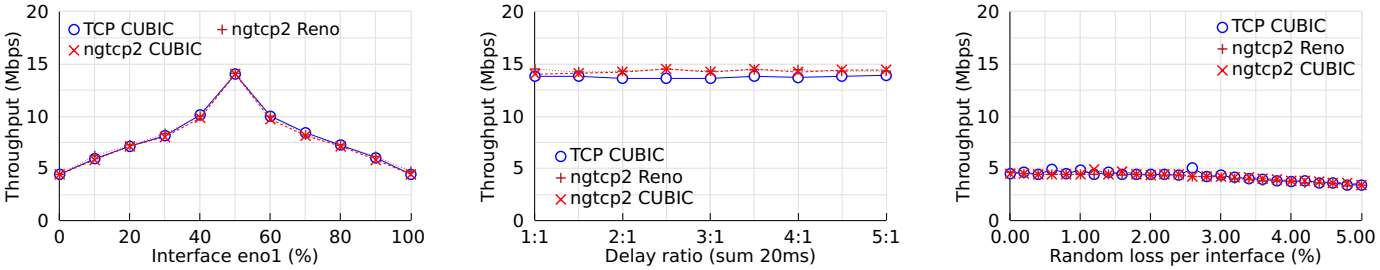
(d) DC batch size, ngtcp2 (repeated Fig. 12(a)) (e) High delay ratio, ngtcp2 (rep. Fig. 13(b)) (f) Random loss, ngtcp2 (repeated Fig. 13(c))

Figure 20: Trace-driven bandwidth variability tests using aioquic and ngtcp2 as QUIC implementations



(a) DC batch split (repeated Fig. 7(a)) (b) Low delay ratio (repeated Fig. 9(a)) (c) Duplicate packets (repeated Fig. 11(a))

Figure 21: Additional trace-based throughput validation tests using aioquic as QUIC implementation



(a) DC batch split (repeated Fig. 13(a)) (b) Low delay ratio (repeated Fig. 18(a)) (c) Duplicate packets (repeated Fig. 19(a))

Figure 22: Additional trace-based throughput validation tests using ngtcp2 as QUIC implementation

larger batch size and an uneven split due to reduced link utilization. With a large batch size, we see significant periods where one link is underutilized and idle while packets are forwarded through the second link. We achieve the highest throughput using a batch size near 100-150 and an even batch split of 50/50 to send half of the packets over each interface. When varying the bandwidth ratio, we observe the highest throughput when the two links have a similar capacity and that the throughput decreases with an uneven link capacity. However, we obtain high throughput for all ratios if the DC ratio is selected to match the bandwidth ratio. When studying the delay, we observe the highest throughput achieved with a ratio of 1:1. For other ratios, the throughput is only negatively impacted in the case of having a high delay. Finally, as expected, higher loss results in lower throughput. However, when using DC to duplicate packets, the throughput is much more stable as the packet losses increase.

Fairness: The fairness of competing connections reduces with a large DC batch size as SC clients monopolize the link during DC client's off periods. We observe the highest fairness using a small batch size and an even batch split. High fairness is achieved using a bandwidth ratio of 1:1, but decreases with a more uneven ratio. When the DC ratio is selected to match the bandwidth ratio, the fairness still decreases but at a slower rate. When varying the delay ratio, the fairness is only significantly impacted when having a high delay. For packet losses, the fairness is in general not impacted. However, the fairness

is much affected when using DC packet duplication, showing much unfairness regardless of the loss rate.

Additional experiments: We generally observe similar trends when repeating the experiments using another QUIC implementation (ngtcp2) and congestion control algorithm (CUBIC). The main differences observed are (1) the optimal point can vary slightly due to implementation differences (e.g., pacer), (2) ngtcp2 is more aggressive than aioquic, leading to higher throughput and less fairness, and (3) only small differences are observed between CUBIC and NewReno. Our performance experiments over variable bandwidth links show similar trends, validating our results under the more realistic scenario.

6. Conclusions

In this paper, we present the first performance study of QUIC over DC. Key insights are given for network operators to understand how different DC parameters and network conditions affect QUIC performance. Regardless if we use aioquic or ngtcp2 as the QUIC implementation or whether we use CUBIC or NewReno as the congestion control algorithm, QUIC's throughput is found to be similar to that of TCP in general cases, provided that the UDP receive buffer (when using aioquic) has been increased to a similar size as the corresponding TCP buffer. We show that QUIC can take advantage of DC when the links share similar properties, and the DC batch size is small. When the properties of the links are too far apart, QUIC performance suffers to the degree that the performance would be better if DC was turned off. Furthermore, we show that QUIC can achieve system-wide fairness, provided that the link properties are similar. Otherwise, the DC connection will suffer more than the single connections, showing poor performance and a high degree of unfairness. We also show that packet duplication allows QUIC to improve throughput for lossy environments at the cost of substantially increased unfairness.

With aioquic, the QUIC throughput is considerably lower if the UDP receive buffer remains at default values for Linux, as PDCP introduces packet bursts, causing packet drops due to full buffers. This occurs especially often in asymmetric link scenarios with high throughput. With the increased use of QUIC, we emphasize the importance of studying and optimizing the resources provided by the kernel to QUIC.

As QUIC is implemented in user-space, interesting future work include studying CPU usage and NIC offloading when used in conjunction with DC. Other future work include mathematical modeling and evaluation of the performance of using QUIC over DC. Here, we take only an experimental approach.

Acknowledgement

The authors are very thankful to Stefan Sundkvist (at Ericsson) for his feedback and help with this work. This work was funded in part by the Swedish Research Council (VR) and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] D. Hasselquist, C. Lindstrom, N. Korzhitskii, N. Carlsson, A. Gurtov, QUIC Throughput and Fairness over Dual Connectivity, in: Proc. IEEE Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) Workshop, 2020.

[2] S. Alfredsson, A. Brunstrom, M. Sternad, Cross-layer analysis of TCP performance in a 4G system, in: Proc. of SoftCOM, 2007.

[3] I. Da Silva, G. Mildh, J. Rune, P. Wallentin, J. Vikberg, P. Schliwa-Bertling, R. Fan, Tight integration of new 5G air interface and LTE to fulfill 5G requirements, in: Proc. of VTC Spring, 2015.

[4] N. H. Mahmood, M. Lopez, D. Laselva, K. Pedersen, G. Berardinelli, Reliability Oriented Dual Connectivity for URLLC services in 5G New Radio, in: Proc. of ISWCS, 2018.

[5] 3GPP, Summary of Rel-15, Tech. Rep. 21.915 Release 15, 2019.

[6] Q. De Coninck, O. Bonaventure, Multipath QUIC: Design and Evaluation, in: Proc. of ACM CoNEXT, 2017.

[7] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, Design, Implementation and Evaluation of Congestion Control for Multipath TCP, in: Proc. of USENIX Symposium on NSDI, 2011.

[8] J. R. Iyengar, P. D. Amer, R. Stewart, Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths, IEEE/ACM Trans. on Networking (2006).

[9] 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, Z. Shi, The QUIC Transport Protocol: Design and Internet-Scale Deployment, in: Proc. of ACM SIGCOMM, 2017.

[10] IETF 106 Singapore, Some updates on QUIC deployment numbers, 2019. URL: `https://datatracker.ietf.org/meeting/106/materials/slides-106-maprg-quic-deployment-update`.

[11] 3GPP, Study on Small Cell enhancements for E-UTRA and E-UTRAN; Higher layer aspects, Tech. Rep. 36.842 Release 12, 2013.

[12] A. Ravanshid, P. Rost, D. S. Michalopoulos, V. V. Phan, H. Bakker, D. Aziz, S. Tayade, H. D. Schotten, S. Wong, O. Holland, Multi-connectivity functional architectures in 5G, in: Proc. of IEEE ICC, 2016.

[13] 3GPP, Evolved Universal Terrestrial Radio Access; Packet Data Convergence Protocol specification, Tech. Rep. 36.323 Release 16, 2020.

[14] M. Polese, M. Mezzavilla, S. Rangan, M. Zorzi, Mobility Management for TCP in MmWave Networks, in: Proc. ACM mmNets, 2017.

[15] B. Jin, S. Kim, D. Yun, H. Lee, W. Kim, Y. Yi, Aggregating LTE and Wi-Fi: Toward Intra-Cell Fairness and High TCP Performance, IEEE Trans. on Wireless Communications (2017).

[16] Y. Khadraoui, X. Lagrange, A. Gravey, TCP Performance for Practical Implementation of Very Tight Coupling between LTE and WiFi, in: Proc. of IEEE VTC Fall, 2016.

[17] Y. Wu, Y. He, L. P. Qian, J. Huang, X. Shen, Optimal resource allocations for mobile data offloading via dual-connectivity, IEEE Trans. on Mobile Computing (2018).

[18] Y. Wu, X. Yang, L. P. Qian, H. Zhou, X. Shen, M. K. Awad, Optimal dual-connectivity traffic offloading in energy-harvesting small-cell networks, IEEE Trans. on Green Communications and Networking (2018).

[19] L. Sharma, B. B. Kumar, S.-L. Wu, Performance analysis and adaptive DRX scheme for dual connectivity, IEEE Internet of Things Journal (2019).

[20] M. He, C. Hua, W. Xu, P. Gu, X. S. Shen, Delay optimal concurrent transmissions with raptor codes in dual connectivity networks, IEEE Trans. on Network Science and Engineering (2021).

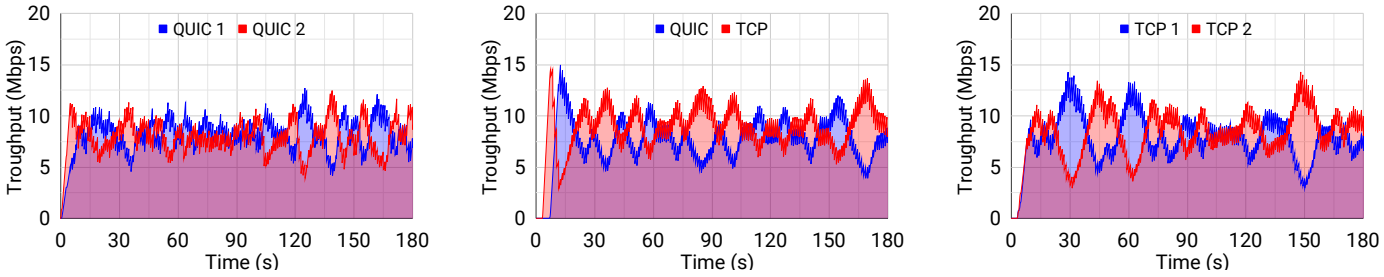
[21] A. Gurtov, T. Polishchuk, Secure multipath transport for legacy Internet applications, in: Proc. of IEEE Broadnets, 2009.

[22] R. S. Mogensen, C. Markmoller, T. K. Madsen, T. Kolding, G. Pocovi, M. Lauridsen, Selective Redundant MP-QUIC for 5G Mission Critical Wireless Applications, in: Proc. of IEEE VTC Spring, 2019.

[23] A. Rabitsch, P. Hurtig, A. Brunstrom, A Stream-Aware Multipath QUIC Scheduler for Heterogeneous Paths, in: Proc. of ACM SIGCOMM workshop EPIQ, 2018.

[24] M. Becke, T. Dreibholz, H. Adhari, E. P. Rathgeb, On the fairness of transport protocols in a multi-path environment, in: Proc. of IEEE ICC, 2012.

[25] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, M. Handley, Data center networking with multipath TCP, in: Proc. of ACM SIGCOMM workshop HotNets, 2010.



(a) QUIC vs. QUIC (JFI=1.000) (b) QUIC vs. TCP (JFI=0.995) (c) TCP vs. TCP (JFI=0.999)

Figure 23: Pairwise example traces using SC

[26] X. Zhang, B. Li, Dice: A Game Theoretic Framework for Wireless Multipath Network Coding, in: Proc. of ACM MobiHoc, 2008.

[27] R. K. Jain, D.-M. W. Chiu, W. R. Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, Technical Report DEC-TR-301, Eastern Research Lab, Digital Equipment Corporation, 1984.

[28] aioquic, aioquic, 2020. URL: `https://github.com/aiortc/aioquic`.

[29] C. Paasch, R. Khalili, O. Bonaventure, On the Benefits of Applying Experimental Design to Improve Multipath TCP, in: Proc. of ACM CoNEXT, 2013.

[30] R. Marx, W. Lamotte, J. Reynders, K. Pittevils, P. Quax, Towards QUIC Debuggability, in: Proc. of ACM SIGCOMM workshop EPIQ, 2018.

[31] R. Marx, M. Piraux, P. Quax, W. Lamotte, Debugging QUIC and HTTP/3 with qlog and qvis, in: Proc. of Applied Networking Research Workshop, 2020.

[32] K. L. McMillan, L. D. Zuck, Formal Specification and Testing of QUIC, in: Proc. of ACM SIGCOMM, 2019.

[33] R. Marx, J. Herbots, W. Lamotte, P. Quax, Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity, in: Proc. of ACM SIGCOMM workshop EPIQ, 2020.

[34] ngtcp2, ngtcp2, 2020. URL: `https://github.com/ngtcp2/ngtcp2`.

[35] D. Raca, J. J. Quinlan, A. H. Zahran, C. J. Sreenan, Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics, in: Proc. of ACM MMSys, 2018.

[36] J. Iyengar, I. Swett, QUIC Loss Detection and Congestion Control, Internet-Draft draft-ietf-quic-recovery-29, IETF, 2020.

Appendix A. Pairwise fairness example using SC

Figure 23 shows throughput example traces with two competing SC clients using the following three configuration pairs: (a) QUIC vs. QUIC, (b) QUIC vs. TCP, and (c) TCP vs. TCP. While the fairness index in all three cases are close to one (i.e., optimal fairness), the differences in the time-variation between the different connections are smallest for the two competing QUIC flows (i.e., Figure 23(a)). One contributing factor to this difference may be the QUIC pacer smoothing out the saw-tooth behavior of NewReno.