QUIC Delay Control: an Implementation of Congestion and Delay Control

Saverio Mascolo, Andrea Vittorio Balillo, Gioacchino Manfredi, Davide D'Agostino, Luca De Cicco
Politecnico di Bari, Bari, Italy
Emails: name.surname@poliba.it

***Abstract*—A new congestion and delay control algorithm named QUIC-Delay Control (QUIC-DC) is proposed for controlling not only congestion but also the queueing delay encountered along the forward communication path. The core idea is to estimate the one-way queueing delay of a connection to trigger an early reaction to congestion. This idea, along with the TCP Westwood+ congestion control algorithm, has been implemented in QUIC-DC and compared with QUIC Cubic, BBRv2, NewReno, Westwood+. The results obtained in the emulated and real network environments show that QUIC-DC can significantly reduce packet losses along with end-to-end communication delays, while preserving network utilization, features that are both very useful for real-time applications.**

***Index Terms*—Congestion Control, Delay Control, QUIC**

I. Introduction

Classic congestion control aims at maximizing throughput while minimizing congestion, i.e. packet losses and retransmissions. This implies that communication delays are not controlled and oscillate between a minimum, equal to the propagation time, up to a maximum equal to the propagation time plus the maximum queuing delay encountered along the communication path.

The goal of this paper is to propose an algorithm that not only avoids congestion while still providing high network utilization but also controls the network queueing delay.

Controlling delays is nowadays of the utmost importance since the Internet is not only aimed at delay insensitive data traffic, but is increasingly used for real-time applications such as live streaming, video-conferencing, virtual reality/augmented reality, 360° video, autonomous driving, telerobotics and tele-surgery.

In this paper, we propose a new congestion and delay control and we carry out an experimental evaluation both in controlled and uncontrolled network environments.

Figure 1 shows an Internet connection between two TCP/UDP peers over an IP network. The essential elements that play a key role are the communication links and the buffers. Indeed, TCP/IP or UDP/IP packets go through a set of communication links and buffers connecting the routers that are encountered as they travel from the sender to the destination. Figure 1 shows the sequence of communication links and buffers, each link i with a propagation delay T_{p_i} and each buffer i with a queueing delay T_{q_i}, which are encountered by a packet travelling from the sender to the destination.

The Round Trip Time (RTT) is the sum of the propagation and queueing delays encountered going from the sender to the destination and then back to the sender: $RTT = \sum_{i \in P}(T_{p_i} + T_{q_i})$, where P is the set of links and buffers belonging to the bidirectional communication path.

The propagation delays T_{p_i} are fixed and depend on the physical distance (the propagation speed is roughly 1/3 of the speed of light). In contrast, the queueing delays T_{q_i} depend on queueing occupancy and can be reduced by controlling the queue length. The overall queueing delay is $T_q = \sum_{i \in P} T_{q_i}$, whereas the overall propagation delay is $T_p = \sum_{i \in P} T_{p_i}$.

In other terms, the RTT is made of a constant propagation time plus a time-varying stochastic component due to the queueing delay as follows: $RTT = T_p + T_q$.

QUIC-DC is designed to bound the component of delay due to the queueing. Conceptually, it can be viewed as a form of Explicit Congestion Notification (ECN) [1] implemented end-to-end. QUIC-DC has been implemented into the Meta's `mvfst` QUIC implementation and an experimental comparison with QUIC Cubic, BBRv2, NewReno, Westwood+ has been carried out. The obtained results show that QUIC-DC provides lower RTT values, while also reducing the packet loss rate by an order of magnitude compared to the other congestion control variants.

II. Related Work

Traditional loss-based TCP [2], [3] is not suitable for delay-sensitive traffic, such as in the case of video conferencing traffic, because its congestion control mechanism continuously probes for available network bandwidth, creating a periodic pattern in which network queues are cyclically filled up to packet loss and then drained. These queue oscillations introduce a time-varying stochastic queueing delay component, which adds to the fixed network propagation delay, thereby impairing delay-sensitive communications. But more than anything, it is the retransmission of lost packets that harms real-time communication. Therefore, reducing the packet loss rate is fundamental in real-time traffic. For this reason, the idea of using delay measurements to proactively react to the onset of network congestion has been proposed [4]. Issues related to delay measurements and throughput degradation when loss-based flows share the bottleneck with delay-based flows have been addressed in [5]–[7]. The first efforts focusing

This work has been funded by the "LOREN" Project, n. 20223Y85JN, under the PRIN (Progetti di Ricerca d'Interesse Nazionale) 2022 programme of the Italian Ministry of Research.

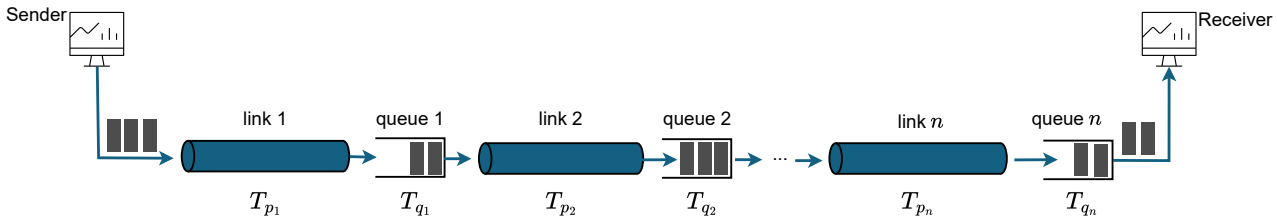


Fig. 1. Packets going from the sender to the receiver through a series of links and buffers

on the reduction of the queueing delay were set in the TCP congestion control research domain. In particular, the first congestion control algorithm specifically designed to contain the end-to-end latency by employing delay measurements is described in the seminal work by Jain [4]. Since then, many delay-based TCP congestion control variants have been designed, such as TCP Vegas [8] or [6]. TCP Vegas employs end-to-end RTT measurements to establish delay thresholds used to infer incipient congestion by comparison with a base RTT [8]. It has been shown that, when the RTT is used as a congestion metric, a low channel utilization may be obtained in the presence of reverse traffic, which inflates queues in the backward path, or when competing with loss-based flows [5]. To tackle such fairness issues, the algorithms proposed in [6] infer the presence of concurrent loss-based flows and switch to loss-based mode, returning back to the delay-based mode when loss-based flows are detected to be no longer present. Other algorithms employ one-way-delay measurements to rule out the sensitivity to reverse-path congestion. Examples are LEDBAT (over UDP) [9] and TCP Santa Cruz [10]. In particular, LEDBAT employs an increasing phase in which the congestion window is increased at a rate that is proportional to the distance between the measured one-way delay and a fixed delay target [9]. It has been shown that LEDBAT is affected by the so-called *latecomer effect*: when two flows share the same bottleneck the second flow typically starves the first one [11]. The idea of employing RTT gradient has been proposed to overcome the aforementioned latecomer effect. Some examples are CDG [12] and Verus [13]. CDG has been designed with the aim of coexisting with loss-based flows while keeping end-to-end delay low [12]. Verus has been designed for the case of cellular networks where sudden link capacity variations make the congestion control design challenging [13]. The Google Congestion Control (GCC) [14], implemented in the Google Chrome web browser, employs the one way delay variation as a congestion signal and an adaptive threshold mechanism to effectively compete with loss-based congestion control algorithms [14].

Recently, several papers have studied the performance of QUIC congestion control algorithms in mobile and satellite networks. In [15] an experimental comparison of congestion control algorithms (CCAs) in 5G networks was carried out using QUIC. It has been shown that Cubic and BBR suffer from significant bufferbloat, while Copa experiences reduced throughput and higher delays. When the bottleneck is primarily in the 5G link, the cellular-optimized RBBR variant demonstrates substantial delay improvements over both BBR and Cubic, highlighting the need for 5G-specific adaptations in congestion control design. In [16] QUIC performance is evaluated over StarLink satellite connections. The paper exploits Starlink's predictable handover events occurring every 15 seconds to propose a congestion control congestion control-agnostic mechanism that prevents unnecessary congestion window reductions. Implementation with BBRv3 and CUBIC in QUIC demonstrated performance improvements up to 35% in both Mininet emulations and real-world Starlink networks, demonstrating the benefits of incorporating handover awareness into CC algorithms for satellite communications.

III. QUIC Delay Control

In this section, we introduce the main idea to measure queueing delay control. The algorithm will be implemented and evaluated in QUIC [17].

To focus on the delay problem, let us consider again Figure 1, which shows a connection between two TCP/UDP peers over an IP network. As we have seen, the RTT is the sum of the encountered propagation delays and queueing delays.

Therefore, by keeping the queue as low as possible, we can target a minimum RTT equal to the propagation delay only:

$$RTT_{\text{min}} = T_p \tag{1}$$

Similarly, RTT_{max} can be defined as:

$$RTT_{\text{max}} = RTT_{\text{min}} + T_{q_{\text{max}}} \tag{2}$$

where $T_{q_{\text{max}}}$ is the maximum queueing time encountered by a packet going through all the buffers belonging to the communication path.

At this point, it is important to distinguish between queueing on the forward path and queueing on the backward path. A connection is primarily constrained by congestion on the forward path; therefore, if RTT is used as a congestion signal, queueing in the backward path caused by reverse traffic may lead to a false indication of congestion. This, in turn, would unnecessarily slow down the forward traffic — a behavior that is indeed observed in TCP Vegas [5].

To this end, it is necessary to consider the variation of the One-Way Delay (OWD), where the OWD is the time a

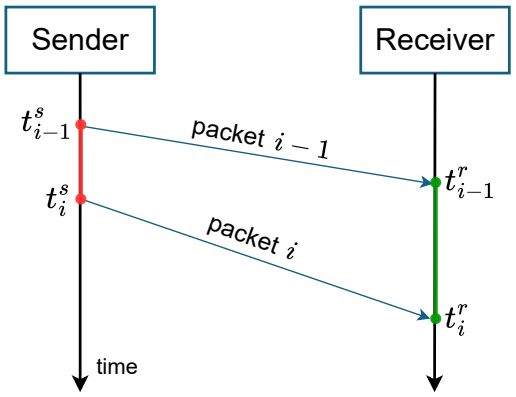


Fig. 2. Measurement of one-way delay variation

packet takes to travel from the sender to the receiver, in order to estimate the forward queueing delay only. Precisely, the OWD is the difference between the packet arrival time t^r at the receiver and the packet departure time t^s at the sender as shown in Figure 2:

$$OWD_i = t^r_i - t^s_i \tag{3}$$

The OWD measurement contains a systematic error due to the clock offset between the sender and the receiver. However, what is useful to know is not the absolute value but just the OWD variation $OWDV$, which measures an increase or decrease in the *forward queueing delay*. Thus, by computing the *one-way delay variation*:

$$\begin{aligned} OWDV_i &= OWD_i - OWD_{i-1} \\ &= (t^r_i - t^s_i) - (t^r_{i-1} - t^s_{i-1}) \\ &= (t^r_i - t^r_{i-1}) - (t^s_i - t^s_{i-1}) \end{aligned} \tag{4}$$

Equation 4 shows that the *one-way delay variation* can be easily obtained by computing the difference between the inter arrival times of two consecutive packets i and i–1, and their corresponding inter departure times.

By summing the one way delay variations, we can compute the one-way queueing delay as:

$$OWQD_i = OWQD_{i-1} + OWDV_i \tag{5}$$

It should be noted that this measure is both simple and robust and can be easily implemented.

Once we have the measurement of the queueing delay along the forward path, we can use it to define *a new congestion event* which occurs when $OWQD$ reaches a threshold $OWQD_{th}$:

$$OWQD > OWQD_{th} \tag{6}$$

Using Eq. (6), we can control queue build-up and reduce the risk of packet loss, which is particularly important for latency-sensitive applications.

In QUIC Delay Control (QUIC-DC), we use the TCP Westwood+ congestion control [5], [18] i.e., after a congestion episode, QUIC-DC sets the congestion window equal to the product of the available bandwidth times the minimum RTT, which keeps full the propagation pipe and empty the queueing pipe.

In summary, a congestion episode is defined by a timeout, a 3DUPACK, or OWD $>$ OWD$_{th}$. The reaction to a congestion episode is the same of that of TCP Westwood+ after a timeout or after 3DUPACK, and the reaction to OWD $>$ OWD$_{th}$ is the same of TCP Westwood+ after 3DUPACK.

The implementation of TCP Westwood+ congestion control [5] in QUIC-DC closely adheres to its counterpart implementation in the Linux TCP kernel with a small change in the Westwood+ low-pass filter to make the bandwidth estimation BWE faster. Indeed, the original low pass filter used to estimate the available bandwidth BWE adversely impacts the algorithm's responsiveness during transient network conditions, leading to a lower goodput during the start up phase. The low pass filter employed in QUIC-DC is:

$$BWE_i = 0.2 \cdot BWE_{i-1} + 0.8 \cdot bandwidthsample_i \tag{7}$$

which significantly improves the goodput.

IV. QUIC-DC Implementation and the Testbed

QUIC-DC has been implemented into Meta's `mvfst` QUIC transport implementation, which serves as the underlying transport layer for `Proxygen`, Meta's HTTP libraries collection[1]. Both `mvfst` and `Proxygen` are implemented in C++ to ensure high performance and scalability. To simplify deployment and testing, all components were containerized using Docker. In our laboratory testbed, we have set the maximum possible value for the advertised window so that the congestion window size is constrained only by the congestion window.

QUIC-DC has been evaluated using two testbeds. The first is a controlled laboratory testbed made of two machines connected with an Ethernet cable. The first machine acts as the client, sending one or multiple HTTP3/QUIC requests to the second machine, which acts as the server. Both machines are equipped with Gigabit Ethernet Network Interface Controllers (NICs). All the Docker containers are based on Ubuntu 22.04 images. The bottleneck is located physically on the interface of the server where we applied a 25 ms delay both incoming and outgoing, thus summing up to a total propagation time of 50 ms. We use the Linux traffic control (`tc`) tool from the `iproute2` suite to perform all the network emulation tasks[2]. The propagation delay T_p is set using `netem` queueing disciplines on the server Ethernet interface, while the bandwidth capacity C in Mbps and the buffer size B in bytes are set using the token bucket filter (TBF) mechanism. The results have been averaged over 3 runs and appear consistent between each run.

[1]https://engineering.fb.com/2014/11/05/production-engineering/introducing-proxygen-facebook-s-c-http-framework

[2]https://man7.org/linux/man-pages/man8/tc.8.html

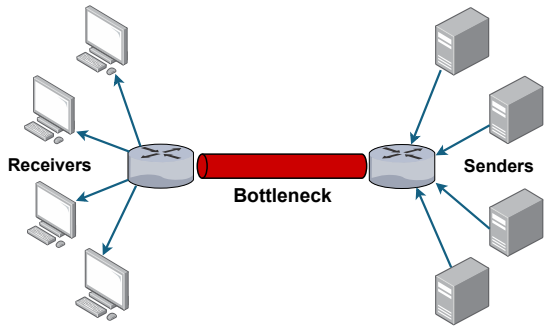


Fig. 3. Single bottleneck testbed

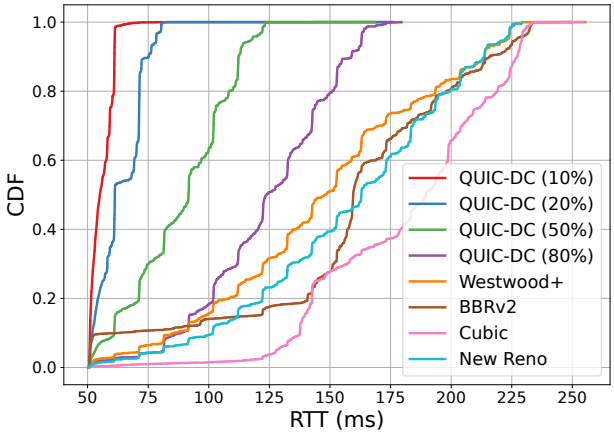


Fig. 4. RTT distribution for a single flow – emulated link, buffer=4×BDP

The other testbed considered is an uncontrolled real Internet setup consisting of a residential client in Bari that accesses the Internet using an ADSL modem. The server is located at the University of Rome, La Sapienza.

Pacing is enabled at the application level to ensure that packets are transmitted at evenly spaced intervals, minimizing burstiness. [3]

V. Experimental results

QUIC-DC has been compared with other state-of-the-art congestion control algorithms, namely Westwood+ [5], [18], Cubic [19], New Reno [20], and BBR version 2 [21].In this work BBRv2 has been considered due to its superior fairness compared to BBRv3 [22]. The bottleneck buffer size B has been set equal 0.5, 1, 2, 4 times the bandwidth delay product (BDP), where the bandwidth is the bottleneck capacity C and the delay is the round trip propagation delay $T_p = RTT_{min}$. We will consider single-flow and multiple-flows scenarios.

A. Emulated network results

1) Single Flow: To understand the dynamics of QUIC-DC, we begin by examining a single-flow connection transferring a 100 MB file. QUIC-DC has been tested using different values for the one-way queueing delay threshold $OWQD_{th}$, which has been set equal to 10%, 20%, 50%, and 80% of the bottleneck buffer size expressed in time.

Table I shows the performance metrics of a single flow connections for the case of a buffer size set equal to 4 BDP, where C=10 Mbps and $RTT_{min} = T_p$ =50 ms. In this case, all the congestion control algorithms achieve comparable goodputs but with larger loss rate in the case of Cubic and BBRv2. In comparison to QUIC Cubic, the loss percentage is reduced up to two orders of magnitude.

Figure 4 shows the corresponding CDF of the measured RTT for all the algorithms under analysis. The curves show that QUIC-DC provides lower RTT values compared to the other algorithms. Moreover, it can be noticed that by decreasing the $OWQD_{th}$ the RTT curves shift to the left towards lower RTT values. This is expected since a higher $OWQD_{th}$ implies larger queues along the path. It is worth noting that in the case of algorithms with a larger loss rate, the RTT measurements underestimate the effective delays because retransmitted packets experience very large delays, which do not contribute to the RTT measurements [23]. Similar considerations can be drawn from Table II, which corresponds to the case of bottleneck buffer size set to 2 BDP.

TABLE I
Single-flow emulation; buffer = 4 × BDP ($C = 10$ Mbps, $RTT_{\min} = 50$ ms).

CCA	Gput (Mbps)	Tput (Mbps)	Loss (%)	RTT_{avg} (ms)	RTT_{std} (ms)
QUIC-DC (10%)	9.16	9.16	0.02	**55.75**	**3.00**
QUIC-DC (20%)	9.17	9.17	**0.01**	63.83	7.64
QUIC-DC (50%)	9.16	9.17	0.03	88.60	19.14
QUIC-DC (80%)	**9.16**	9.16	0.02	123.82	26.48
Westwood+	9.18	9.19	0.07	170.23	43.10
BBRv2	9.15	9.21	0.65	102.33	27.44
Cubic	9.10	9.20	1.03	190.32	31.37
New Reno	9.20	9.20	0.07	159.26	45.14

A nice understanding of the meaning of the proposed approach is provided by Figure 5, which compares the RTT and $cwnd$ dynamics of QUIC Westwood+ (blue line) and QUIC-DC (red line) in the case of a bottleneck buffer size set to 2 BDP. It can be observed that, while QUIC Westwood+ resets the congestion window when the maximum RTT is reached (dashed green line), QUIC-DC resets the congestion window when the $OWQD$ reaches the delay control threshold set at 80% of the bottleneck buffer size in time. Under this setting, buffer saturation at the bottleneck is prevented, effectively avoiding packet losses. Consequently, the observed packet loss ratio was reduced by one order of magnitude (see Table I).

When the buffer size is set to 1 BDP or 0.5 BDP, the results in Table III and Table IV indicate that QUIC-DC tends to be less efficient in terms of goodput. Notably, in this scenario with an undersized bottleneck buffer, both BBRv2 and Cubic achieve higher goodput, with Cubic also exhibiting the lowest loss rate. As will be shown in the following, this result disappears in the case of a multi-flow scenario.

[3]The pacing interval has been set to 200 μs.

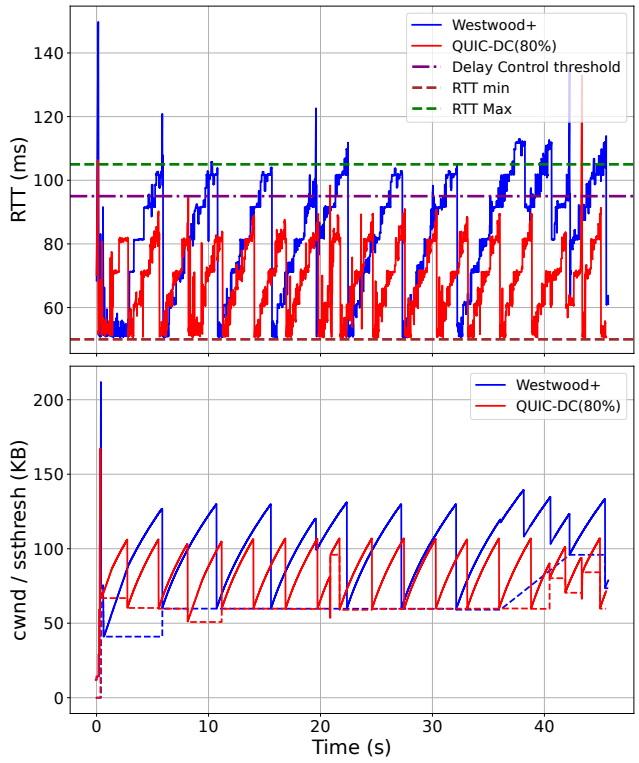


Fig. 5. Time evolution of RTT and congestion window – buffer=2×BDP.

TABLE II
SINGLE-FLOW EMULATION; BUFFER = 2 × BDP ($C = 10$ MBPS, $RTT_{\min} = 50$ MS).

CCA	Gput (Mbps)	Tput (Mbps)	Loss (%)	RTT_{avg} (ms)	RTT_{std} (ms)
QUIC-DC (10%)	9.06	9.06	0.03	**52.74**	1.23
QUIC-DC (20%)	9.16	9.16	0.03	55.98	3.65
QUIC-DC (50%)	9.14	9.15	0.14	68.33	10.25
QUIC-DC (80%)	**9.16**	9.16	**0.04**	84.86	16.47
Westwood+	9.15	9.18	0.26	90.53	20.04
BBRv2	9.12	9.18	0.66	87.94	26.94
Cubic	9.16	9.20	0.39	104.82	13.57
New Reno	9.17	9.18	0.14	93.78	18.79

TABLE III
SINGLE-FLOW EMULATION; BUFFER = 1 × BDP ($C = 10$ MBPS, $RTT_{\min} = 50$ MS).

CCA	Gput (Mbps)	Tput (Mbps)	Loss (%)	RTT_{avg} (ms)	RTT_{std} (ms)
QUIC-DC (10%)	7.08	7.13	0.63	52.51	1.06
QUIC-DC (20%)	7.75	7.79	0.53	52.63	1.65
QUIC-DC (50%)	8.51	8.58	0.83	53.64	3.96
QUIC-DC (80%)	8.39	8.46	0.83	52.44	1.15
Westwood+	8.22	8.27	0.64	53.11	3.03
BBRv2	9.12	9.21	0.87	57.21	7.72
Cubic	9.19	9.20	0.15	63.06	7.15
New Reno	7.39	7.42	0.37	52.39	1.10

TABLE IV
SINGLE-FLOW EMULATED LINK; BUFFER = 0.5 × BDP ($C = 10$ MBPS, $RTT_{\min} = 50$ MS).

CCA	Gput (Mbps)	Tput (Mbps)	Loss (%)	RTT_{avg} (ms)	RTT_{std} (ms)
QUIC-DC (10%)	3.30	3.32	0.44	52.45	1.49
QUIC-DC (20%)	3.58	3.62	1.24	52.23	1.06
QUIC-DC (50%)	3.44	3.49	1.42	52.11	1.02
QUIC-DC (80%)	3.75	3.82	1.77	52.26	1.04
Westwood+	3.68	3.73	1.33	52.29	1.02
BBRv2	6.27	6.35	1.20	51.41	0.77
Cubic	5.53	5.54	0.24	51.32	0.95
New Reno	3.34	3.37	0.81	52.48	0.96

TABLE V
FOUR CONCURRENT FLOWS, EMULATED LINK – BUFFER=2×BDP. (AVERAGE ± STANDARD DEVIATION).

CCA	RTT_{avg} (ms)	Goodput (Mbps, JFI)	Loss (%)
QUIC-DC (80%)	$\mathbf{95.07 \pm 0.16}$	2.29 ± 0.07 (0.999)	$\mathbf{0.12 \pm 0.02}$
Westwood+	101.19 ± 0.13	2.28 ± 0.12 (0.998)	0.43 ± 0.06
BBRv2	111.55 ± 0.29	2.28 ± 0.34 (0.984)	1.05 ± 0.11
Cubic	114.26 ± 0.09	2.30 ± 0.03 (0.999)	0.34 ± 0.03
New Reno	113.87 ± 0.44	1.95 ± 0.12 (0.998)	0.58 ± 0.07

2) Multiple Flows: The multi-flow scenario shown in Figure 3 is considered. In particular, four connections of the same type are started at different times to avoid synchronization. In this case, the control delay threshold $OWQD_{th} = 80\%$ and the bottleneck buffer size is set to 2 BDP. We have measured and averaged goodputs, throughputs, losses and RTTs for each connection.

Table V presents the average values along with the corresponding standard deviations for the RTT, goodput, Jain's fairness index, and loss percentage of each control algorithm. As shown, QUIC-DC exhibits the lowest RTT and loss percentage, while maintaining a goodput level comparable to the other considered algorithms. It is worth noting that, since the link capacity is set to 25 Mbps, a goodput around 6 Mbps indicates that the link is fully utilized and fairly shared among the four connectioncs.

Figure 6 shows the CDF of the RTT in the case of multiple connections over a shared bottleneck. It shows that QUIC-DC provides lower RTT values compared to the other considered

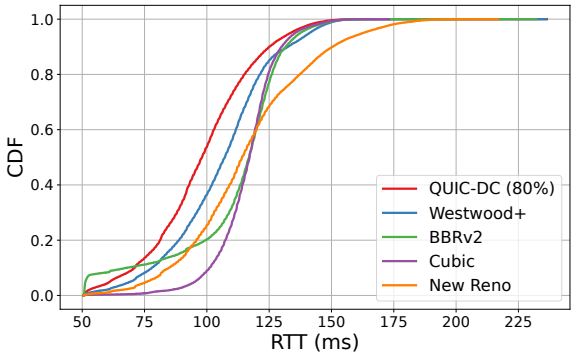


Fig. 6. RTT distribution for four concurrent flows – emulated link, buffer=2×BDP.

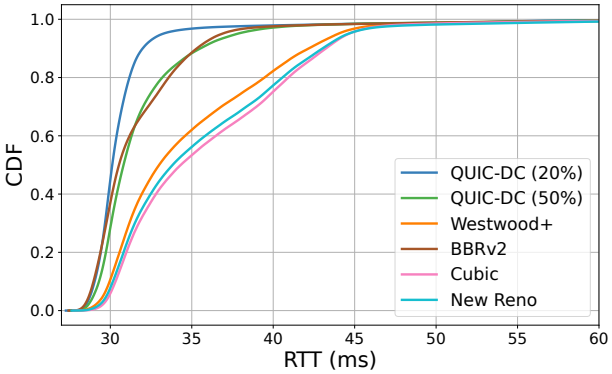


Fig. 7. RTT distribution on real network path.

TABLE VI
SINGLE FLOW, REAL NETWORK CONNECTIONS.

CCA	**Gput** (Mbps)	**Tput** (Mbps)	**Loss** (%)	**RTT**$_{avg}$ (ms)	**RTT**$_{std}$ (ms)
QUIC-DC (20%)	26.744	26.746	0.0096	30.92	9.30
QUIC-DC (50%)	30.954	30.956	0.0103	32.09	11.18
Westwood+	33.115	33.132	0.0470	35.08	11.50
BBRv2	33.560	33.586	0.0730	32.00	11.28
Cubic	33.529	33.555	0.0733	36.17	12.05
New Reno	33.935	32.959	0.0765	35.92	12.18

congestion control algorithms. Due to space constraints, we are here unable to investigate the friendliness between different types of control algorithms.

B. Real network results

We consider the real network scenario described in Section IV. During the experiments, we have measured $RTTmin = 30\ ms$, $RTTmax = 60\ ms$, which means that there is a bottleneck buffer size equal to 1 BDP. Table VI shows that QUIC-DC reduces the loss rate by one order of magnitude whereas the goodput is only slightly lower than the other approaches (note that QUIC Westwood+ corresponds to QUIC-DC (100%)). Moreover, Figure 7 shows that QUIC-DC also provides shorter RTT. However, it should also be noted that algorithms experiencing higher loss rates provide RTT delays that are underestimated because retransmitted packets are not considered in RTT measurements [23]. In other terms, the algorithms with larger loss-rates provide a distribution of RTT that underestimates the effective delay.

VI. CONCLUSIONS

In this paper we have proposed QUIC-DC a novel congestion control algorithm that controls the delay of an Internet connection by measuring the one-way queueing delay to trigger an early reaction to congestion. The idea has been implemented in QUIC using the TCP Westwood+ congestion control and has been evaluated against QUIC NewReno, QUIC Westwood+, QUIC Cubic, and QUIC BBRv2. Experimental results show that the cumulative distribution function of the round-trip time can be shifted to lower values, while the packet loss rate can be reduced by an order of magnitude. BBRv2 has shown drawbacks when multiple flows share the bottleneck.

REFERENCES

[1] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," Internet Requests for Comments, RFC Editor, RFC 3168, 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3168.txt

[2] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM '88*, ser. SIGCOMM '88, 1988, p. 314–329.

[3] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*. Elsevier, 2021.

[4] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 5, p. 56–71, Oct. 1989.

[5] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, p. 25–38, Apr. 2004.

[6] Budzisz, R. Stanojevic, A. Schlote, F. Baker, and R. Shorten, "On the fair coexistence of loss- and delay-based tcp," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1811–1824, 2011.

[7] R. S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Proc. PFLDNet*, vol. 4, 2004.

[8] L. Brakmo and L. Peterson, "Tcp vegas: end to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[9] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (ledbat)," Internet Requests for Comments, RFC Editor, RFC, 12 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6817.txt

[10] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving tcp congestion control over internets with heterogeneous transmission media," in *Proc. ICNP '99*. IEEE, 1999, pp. 213–221.

[11] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for ledbat fairness," in *Proc. IEEE GLOBECOM '10*. IEEE, 2010.

[12] D. A. Hayes and G. Armitage, "Revisiting tcp congestion control using delay gradients," in *International Conference on Research in Networking*. Springer, 2011, pp. 328–341.

[13] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *Proc. ACM SIGCOMM '15*, 2015, pp. 509–522.

[14] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2629–2642, 2017.

[15] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "Performance of QUIC congestion control algorithms in 5G networks," in *Proc. of ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases*, 2022, p. 15–21.

[16] V. Kamel, J. Zhao, D. Li, and J. Pan, "StarQUIC: Tuning Congestion Control Algorithms for QUIC over LEO Satellite Networks," in *Proc. of International Workshop on LEO Networking and Communication*, ser. LEO-NET '24, 2024, p. 43–48.

[17] J. Iyengar and M. Thomson, "Quic: A udp-based multiplexed and secure transport," Internet Requests for Comments, RFC Editor, RFC 9000, 5 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9000.txt

[18] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "Tcp westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001, pp. 287–297.

[19] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[20] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm," Internet Requests for Comments, RFC Editor, RFC 6582, 2024. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6582.txt

[21] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, I. Swett, V. Vasiliev, B. Wu, M. Mathis, and V. Jacobson, "BBR v2: a model-based congestion control IETF 105 update," *Presentation at IETF105*, 2019.

[22] D. Zeynali, E. N. Weyulu, S. Fathalli, B. Chandrasekaran, and A. Feldmann, "Promises and potential of mv3," in *Proc. of PAM 2024*, 2024, p. 249–272.

[23] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *ACM SIGCOMM Computer Communication Review*, vol. 17, no. 5, pp. 2–7, 1987.