Modified QUIC protocol with congestion control for improved network performance

Prashant Kharat1 | Muralidhar Kulkarni2

1 Department of Information Technology, Walchand College of Engineering, Sangli, India
2 Department of E&C Engineering, NITK, Surathkal, India

Correspondence
Muralidhar Kulkarni, Department of E, and C Engineering, NITK, Surathkal, India.
Email: prashant.kharat@walchandsangli.ac.in

Abstract
In the networks, a transport layer is responsible for reliable data with guaranteed Quality of Service. A Modified-QUIC protocol provides improved throughput and reduced latency of the network. This work proposes a modification in handshaking mechanism for QUIC protocol to minimize overhead due to control signals and time required to update the congestion-window size. This modification fine-tunes the window update mechanism with the acknowledgment frame, which results in a smooth variation in the congestion-window size. This leads to the control congestion by regulating traffic in the network. In the proposed mechanism, unnecessary time out events are avoided by updating the congestion-window on receipt of the acknowledgment frame. This work has been carried out using two different testbed setups to verify the transport layer and a browser network performance. It has been observed that a Modified-QUIC protocol is easy to deploy, improves the throughput and data rate by 35% and 3.43%, respectively, over the QUIC protocol. The average fairness index is increased with a file size and for the long live traffic.

1 INTRODUCTION

To improve the network throughput and to challenge the Transmission Control Protocol’s (TCP’s) dominance in the networking industry, Google group has developed a Quick UDP Internet Connections (QUIC) protocol. In the network, TCP provides guaranteed Quality of Service (QoS) with the few service level limitations. The QUIC has been developed to improve the network performance and overcome TCP’s limitations. The Cisco Visual Networking Index estimated that Annual Global IP traffic will hit 3.3 ZB by 2021. The report states that up to 83% of the Internet Protocol (IP) traffic of consumers is consumed by streaming video and 13% by the live streaming [1]. According to the above studies, to handle fast growing IP traffic, networking protocols are playing a very important role. Due to the increased Internet usage and type of data traffic, Hypertext Transfer Protocol (HTTP) published with RFC-2616 in 1999 is outdated at present. Google has also developed SPDY, a web transfer protocol to improve Page Load Time (PLT) by using techniques such as multiplexed TCP streams [2], and header compression on top of HTTP/1.1.

In 2015, the Internet Engineering Task Force (IETF) published HTTP/2 as RFC-7540 [3]. HTTP/2 is an application layer protocol running on the top of TCP. A three-way handshake in the QUIC increases the Round Trip Time (RTT) of HTTP request and degrades performance further in the presence of TLS encryption. An alternate protocol to TCP is the User Datagram Protocol (UDP), a connection-less and unreliable protocol, which degrade QoS. Google proposed QUIC protocol in 2013 in which UDP is a base protocol and an alternative to HTTP/2 over a TCP/IP [4].

The motivation behind this study is that in today’s fast paced world, the rate and quality of information has ascended to paramount importance. The Cisco survey estimated that by 2021, the annual global IP traffic will hit 3.3 ZB, of which 63% will be shared by wireless and mobile traffic. In this scenario, it becomes ever more relevant to establish networking protocols to handle the rapid growth in the traffic usage. The popular HTTP has become outdated with the changes in Internet usage and traffic. To maximize bandwidth utilization and data delivery rate, UDP protocol can be one of the options, whereas on the contrary TCP limits data rate by adding reliability as the key feature.
In the literature, sufficient works to improve the performance of transport layer and web are available, but most of those are focusing on TCP or HTTP or SPDY [2, 5–8]. QUIC has been proposed recently to compete with the TCP. Before this, there was one attempt made by Vernersson with UDP based reliable transport [9].

This paper is an extended version of our previous preliminary work published in [10] and contributes towards the reduced latency and congestion-free network demands of current Internet users. Google’s QUIC protocol deals with the same and is growing very fast. The QUIC protocol source code is open for researchers to investigate the performance. This work specifically contributes to congestion control and reduced packet transmission latency. To evaluate the QUIC protocol performance and modify the source code, we faced the following key challenges.

1. To evaluate performance, it is often required to know the protocol specification and working model.
2. To identify the parameters that fairly compare performance with other transport layer protocols.
3. Even though the QUIC source code is publicly available, it has to be validated with the actual deployed code.
4. A QUIC source code is a complicated structure. Hence, to locating, modifying and compiling code is a tedious task.

To evaluate the performance of ModQUIC, two testbed setups were prepared using the OpenFlow Mininet platform and Chromium server-client model. The test cases are created by varying the load in terms of the number of packets, bandwidth, and loss rate. In the comparative analysis for critical situations in the network, a ModQUIC protocol performance is better compared to QUIC, TCP and TCP/HTTP/2.

The rest of the paper is organized as follows. Section 2 describes QUIC with its features, congestion control mechanism used and related work. Section 3 elaborates window size update mechanism with mathematical illustration and developed algorithm. A proposed work is explained in Section 4. Section 5 gives detailed information about the experimental setup used and results are presented in Section 6. Finally, the work is summarized and concluded in Section 7.

2 | BACKGROUND OF THE WORK

This section presents an introduction to the QUIC protocol, CUBIC congestion control mechanism, and the literature survey on experimental work carried out with the QUIC protocol.

2.1 | QUIC protocol

The QUIC protocol is a secure, reliable, multiplexed transport on top of the UDP and has the open-source deployment [4, 11]. This section provides information about a QUIC protocol and the existing experimental works related to this study.

2.1.1 | QUIC features

The features that describe the functionality of the QUIC protocol are:

1. Multiplexed streaming: As shown in the Figure 1, QUIC multiplexes different streams over the same UDP connection. QUIC is on the top of UDP, hence out of order delivery is possible, which helps in solving HOL blocking as shown in Figure 2.
2. Less connection establishment latency: The time taken to set up a connection by the QUIC is at the most one RTT and in the case the client has already communicated with the server, it uses zero-RTT as shown in Figure 3. This reduces a connection establishment latency as compared to the traditional TCP. Even in an authentic and secure connection, QUIC (QUIC-Crypto) needs only one RTT, in contrast to TCP+TLS, which needs three RTTs.
3. Authenticated and encrypted header and payload: To secure the data delivery by avoiding a third-party manipulation, QUIC packets are authenticated and the payload part is encrypted. However, if the payload is partially encrypted, it still gets authenticated by the receiver.
4. Stream and connection-level flow control: QUIC consists of connection level (like TCP) and stream level (within connection multiple streams are present) flow control mechanisms. Based on the QUIC receiver capacity, it advertises the absolute bytes of data within each stream or connection (aggregate stream data).
5. Flexible congestion control: QUIC has a flexible/pluggable congestion control mechanism. At present, QUIC has CUBIC [12, 13] and BBR [14] functionalities. Out of these, as default, CUBIC congestion control with packet pacing mechanism is used to handle network traffic. The packet pacing mechanism, shown in Figure 4 is useful to manage busy data. In QUIC, ACK frame supports up to 256 NACK (duplicate ACK in TCP) ranges, so that QUIC withstands more effectively in reordering situation than TCP with Selective-ACK (SACK).
6. Connection migration: QUIC connections are identified by the 64-bit connection ID (instead of four-tuple in TCP), randomly generated by the client. As shown in Figure 5, in case of connection migration, the QUIC connection identification number remains the same throughout the communication time so that it can survive IP address changes and Network Address Translation (NAT) re-bindings. Also, the same session key is used for automatic authentication and cryptographic verification of migrating clients.

2.1.2 | CUBIC: QUIC’s congestion control mechanism

A QUIC has default pluggable congestion control functionality which consists of a CUBIC with a packet pacing mechanism. A CUBIC is an enhanced version of the BIC...
Figure 6 shows the behaviour of CUBIC with respect to the cubic function versus time, for which \( K = \sqrt[3]{\frac{\beta W_{\text{max}}}{C}} \). Whenever a packet dropping event occurs, the \( \text{cwnd} \) is reduced by a factor \( \beta \), otherwise increased by the \( \alpha \) for a successful ACK. For TCP with CUBIC, \( \beta = 0.3 \), whereas for QUIC with CUBIC, \( \beta = 0.3/2 = 0.15 \).

This work suggests \( \beta = \beta_{\text{TCP}}/n \) for QUIC with CUBIC and ModQUIC with CUBIC, where \( n \) is the number of competing flows.

Every new epoch starts at \( t = 0 \) and \( \alpha \) is set to zero. \( W_{\text{max}} \) is the initial \( \text{cwnd} \) size, where packet loss occurred previously. However, Equation (1) preserves properties of the BIC such as RTT fairness, limited slow start, and rapid convergence. As a precaution, CUBIC has a mechanism to ensure that the performance should not get worse than standard Reno by simultaneous checking and calculation of a \( W_{\text{reno}} \) parameter. Through experimental studies, it has been observed that throughput and fairness properties of the CUBIC are very good. As CUBIC is available in the Linux TCP suite (kernel version 2.6.16), at present this is the extensively used congestion control algorithm.

2.2 A survey on experimental work with QUIC protocol

The literature on the QUIC is available as an Internet draft proposed by the researchers from the Google. A secure connection with QUIC-Crypto [16], loss recovery and congestion control [17, 18], QUIC contribution in internet transport [19], QUIC as a test-drive application [20] and QUIC internet draft for HTTP/2 [21, 22].

As the present work is an experimental study of the ModQUIC protocol, this related work section is confined only to the experimental studies that have been undertaken. Most of the QUIC related experimental investigations used to verify the performance is carried out by Google. The Google claimed about 3% improvement using QUIC in the mean PLT compared to TCP [23]. The Google claimed that QUIC can reduce search latency by 8% and 3% for stable and mobile users, respectively [24]. However, it reduces buffering time by 18% for stable and 15.3% for mobile users. In addition to this, Google highlighted features like reduced HoL blocking, improved congestion control, and a loss recovery.
Other than the Google, researchers have explored the QUIC performance for varied scenarios. Carlucci et al. prepared two different experimental setups to investigate the congestion control dynamics and web PLT [25]. They observed that, with a Forward Error Correction (FEC)\(^1\), QUIC goodput\(^2\) performance is very poor. They also noticed that for multiple objects in the presence of losses, QUIC underperforms compared to the TCP/HTTP. Megyesi et al. and Biswal tested the QUIC performance in an emulated environment with QUIC enabled desktop client and a Google server. Both of them observed that the TCP/HTTP outperforms QUIC for a high BDP and more number of the large objects, but there were controversial comments by both these studies in the presence of packet loss [26, 27].

Das in his M.S. work evaluated QUIC performance in Mahimahi, a web performance measurement toolkit. He found that for QUIC the performance was very good for low bandwidth and high RTT links [28]. The efficiency of the QUIC is tested through the various scenarios by Cook et al. using local and remote testbeds. They investigated the QUIC performance for the type of access network, packet loss, and by adding link delay in the network. They observed QUIC outperforms TCP/TLS with HTTP/2 for wireless and mobile networks, whereas for wired and stable networks, a significant performance improvement is not seen [29].

Srivastava in his M.Sc. thesis has compared the performance of QUIC with TCP by using throughput, delay, and fairness parameters. He found that for an added delay and loss, QUIC outperforms and while competing flow QUIC was unfair [30]. Kakhki et al. carried out extensive experimentation for various network conditions [31]. They found that QUIC outperforms TCP/HTTP in almost all scenarios, but QUIC shows

\(^1\) In 2016 due to the lack of significant contribution, FEC functionality is removed from QUIC.

\(^2\) Dividing the amount of application-level data by the total time is taken until the completion of its delivery.
poor performance for the out order delivery. They noticed that QUIC is unfair when competing with the TCP flows to acquire resources.

Duan et al. proposed Intel DPDK based cQUIC: a scalable QUIC transport protocol implemented using Click [32]. They have introduced a modular L2-L3 network stack and claimed that the performance is improved over the Google QUIC server. They have analyzed real traffic and observed that 18% of the QUIC based connections are using 2-RTT handshake by limiting scalability. Coninck and Bonaventure, motivated by the success of Multipath-TCP (MPTCP), proposed Multipath-QUIC (MPQUIC) [33]. They succeeded in enabling the QUIC connection to use different paths. They have presented a comparative analysis of MPQUIC with MPTCP and observed that in lossy scenarios, the MPQUIC performance was seen to be better than MPTCP. Hussein et al. integrated QUIC and SDN for improved resource utilization and network security [34]. They implemented QUIC enabled SDN architecture to maximize bandwidth utilization and to secure data services.

Till today, numerous modifications in TCP have been proposed and presented as the TCP variants. These variants have limitations to enhance data rates, desired for video streaming which is responsible for congestion in the network. This ultimately affects QoS by reducing packet delivery ratio. The encryption technique, TLS, used in the TCP causes jitter effects, which degrade the performance of TCP and modifies TCP connections. In the comparative analysis, we have addressed congestion control issues for which ModQUIC provides better solutions to resolve those.

This work adds a new and extended contribution in terms of a ModQUIC: a modified handshaking mechanism to improve overall throughput and reduce congestion window update delay compared to others which have been summarized in Table 1.

### Table 1: Summary of the literature survey

| Contributors | Version used | Throughput (Goodput) | Loss | Delay | Fairness | PLT | Link Bandwidth | Scenario | Additional Contribution |
|--------------|--------------|----------------------|------|-------|----------|-----|----------------|----------|------------------------|
| Megyesi et al. [26] | 20 | N | Y | N | Y | Y | N | Wired | N |
| Carlucci et al. [25] | 21 | Y | Y | N | N | Y | Y | Wired | N |
| Das [5] | 23 | N | N | N | N | Y | N | Wired | N |
| Biswal and Gnawali [27] | 23 | Y | N | N | N | Y | Y | Wired | N |
| Cook et al. [29] | 25 | N | Y | Y | N | Y | N | Wired/Wireless | N |
| Srivastava [30] | 25 to 36 | Y | Y | Y | N | N | Y | Wired | N |
| Kakhki et al. [31] | 25 to 37 | N | Y | N | Y | Y | Y | Wire/Cellular | N |
| Duan et al. [32] | Not mentioned | Connection setup rate (c/s) | Wired | cQUIC |
| Coninck and Bonaventure [33] | Not mentioned | Y | Y | Y | N | N | N | Wired | MPQUIC |
| Hussein et al. [34] | Not mentioned | Security analysis | Y | Wire/Cellular | N |
| This work | 25 to 39 | Y | Y | Y | Y | N | Y | Wired | ModQUIC |

### Figure 7: Poisson queue transition state

3 | CONGESTION WINDOW GROWTH ANALYSIS

This section deals with the congestion window growth analysis with the help of mathematical illustration and algorithmic approach.

#### 3.1 Mathematical illustration

To determine the flow rate of a node, the probabilities of both the collision and number of packets in the system are calculated. To model this system, a birth-death process is used to handle arrival and departure of the data rates [35]. However, to analyze the system behaviour, a Poisson Distribution Function is used. The queuing model based on the pure death-birth process is shown in Figure 7.

Let $n$ be the number of packets in the network, $\lambda_n$ the arrival rate, $\mu_n$ the departure rate, $P_n$ the steady-state probability of $n$ number of packets, and $P_0$ a function of $\lambda_n$ and $\mu_n$.

These variables are used to determine system performance. Under steady-state condition (i.e. $n > 0$), the expected flow rate into and out of the state are equal.
If the \((n-1)\) to \((n+1)\) changes in the state, then Expected rate, \(R_n\) of the flow into state \(n\) is

\[
R_n = \lambda_{n-1}.P_{n-1} + \mu_{n+1}.P_{n+1}, \tag{3}
\]

Actual rate \(R_n\) of flow into state \(n\) is

\[
R_n = (\lambda_n + \mu_n)P_n. \tag{4}
\]

To achieve maximum matching for expected rates, Equation (3) = Equation (4),

\[
\lambda_{n-1}.P_{n-1} + \mu_{n+1}.P_{n+1} = (\lambda_n + \mu_n)P_n. \tag{5}
\]

Using state transition diagram (Figure 7) for \(n = 0, 1, 2, \ldots\)

For \(n = 0\), Equation (5) becomes

\[
\lambda_{0-1}.P_{0-1} + \mu_{1}.P_1 = (\lambda_0 + \mu_0)P_0, \tag{6}
\]

As \(\lambda_{0-1}, P_{0-1}\) and \(\mu_0\) states are absent, Equation (6) becomes,

\[
\mu_1.P_1 = \lambda_0.P_0, \tag{7}
\]

therefore,

\[
P_1 = \frac{\lambda_0}{\mu_1}.P_0 \tag{8}
\]

for \(n = 1, \)

\[
\lambda_{0}.P_0 + \mu_2.P_2 = (\lambda_1 + \mu_1).P_1. \tag{9}
\]

using \(P_1\) from Equation (8) into Equation (9),

\[
P_2 = \frac{(\lambda_1, \lambda_0)}{(\mu_2, \mu_1)}.P_0, \tag{10}
\]

hence,

\[
P_n = \frac{(\lambda_{n-1}, \lambda_{n-2}, \ldots, \lambda_0)}{(\mu_n, \mu_{n-1}, \ldots, \mu_1)}.P_0, \tag{11}
\]

for \(n = 0, 1, 2, \ldots\)

\[
\sum_{n=0}^{\infty} P(n) = 1. \tag{12}
\]

The congestion is estimated based on the packet emission probability using the previous state. A packet emission probability is calculated using window update information sent by the receiving node. A three step window update mechanism is used at the source end to control the packet transmission rate. Let, \(W_n\) be the window update:

\[
W_n = 0, \text{ if } P_n = P_{n-1}, \tag{13a}
\]

\[
W_n = 1, \text{ if } P_n > P_{n-1}, \tag{13b}
\]

\[
W_n = -1, \text{ if } P_n < P_{n-1}. \tag{13c}
\]

ALGORITHM 1 Window Update

Input \(\text{PACKET}_{\text{old}}, \text{BANDWIDTH}_{\text{old}}\) and \(\text{WINDOWSIZE}_{\text{old}}\)

Output \(\text{PACKET}_{\text{new}}, ST\)

if \(\text{BANDWIDTH}_{\text{new}} < \text{BANDWIDTH}_{\text{old}}\), \(\text{WINDOWSIZE}_{\text{new}} = \text{WINDOWSIZE}_{\text{old}} - ST\) then

end

if \(\text{BANDWIDTH}_{\text{new}} > \text{BANDWIDTH}_{\text{old}}\), \(\text{WINDOWSIZE}_{\text{new}} = \text{WINDOWSIZE}_{\text{old}} + ST\) then

end

if \(\text{WINDOWSIZE}_{\text{old}} == \text{WINDOWSIZE}_{\text{new}}\), \(\text{PACKET}_{\text{old}} = \text{PACKET}_{\text{old}}\) then

end

if \(\text{WINDOWSIZE}_{\text{old}} < \text{WINDOWSIZE}_{\text{new}}\), \(\text{PACKET}_{\text{old}} = \text{PACKET}_{\text{old}} + ST\) then

end

if \(\text{WINDOWSIZE}_{\text{old}} > \text{WINDOWSIZE}_{\text{new}}\), \(\text{PACKET}_{\text{old}} = \text{PACKET}_{\text{old}} - ST\) then

end

The above assumptions lead to the following observations or strategies:

1. Decrease in size of the packet will increase the number of packets in the network to send the same amount of data. This increases the probability of collision and results in congestion window size reduction which leads to the congestion control.

2. A size of packet is depends on the available link bandwidth, that is depends on the window size.

3. The window update contains congestion information which controls the rate of transmission by increasing or decreasing of a size of the congestion window.

4. If flow control is achieved, packet size simultaneously decreases which will lead to an increase in the number of packets according to the state-1. To ensure that the probability of packets emitted is less than the rate of transmission to be controlled.

5. The above strategies are used to control the congestion which improve throughput and reduce latency in the network.

3.2 Window size update algorithm

A strategy to update window size is given in Section 3.1 by using Algorithm 1. The available bandwidth estimation is carried out with the help of an MAC layer. If currently the estimated bandwidth is less than the previous bandwidth, the window size needs to be reduced with a step size of \(ST\). If the current bandwidth is greater than the previous bandwidth, the window size is increased by the step size of \(ST\), where the default step size \(ST\)
The data rate is automatically adjusted according to the updated window size. This fine-tuned mechanism along with the ACK frame will result in the smooth variation of a $cwnd$ size and shows the stable system performance.

4 | PROPOSED SCHEME: ModQUIC

The ModQUIC is a refinement in the existing handshaking mechanism of QUIC protocol to update the size of congestion window. In this as shown in Figure 8, the window update frame is coupled with the ACK frame. Initially, ModQUIC establishes a server-client connection by driving QUIC/Crypto request message. In the initial handshake, both the server and client negotiate the $cwnd$ size. This modification helps to reduce the control overhead and packet transmission delay which improves the overall throughput and reduce the latency in the network. The size of $cwnd$ varies concerning cubic function given in Equation (1).

4.1 | ACK frame structure of QUIC protocol

The ACK frame consists of received and missing packet information to notify the server. As shown in Figure 9, ACK frame structure of a QUIC differs from the TCP-ACK frame [21]. A field NACK is used to specify missing packets, whereas server periodically apprise, not to wait below the given sequence number using stop-waiting frame.

4.2 | Window update frame of QUIC protocol

The window update frame as shown in Figure 10 has been used to inform the peer of an increase in end point's flow control using receive window size [21]. Window updates can be applied for the connection-level or stream-level flow control. Violating the flow control by sending more bytes than the prescribed limit results in the closure of the connection. At present, a window size available in the QUIC is 16 KB, which gradually increases during handshaking by exchanging the window control parameters.

Following are the fields of window update frame.

- Frame Type (8-bit): This field set as 0x004 to notify a window update frame.
Stream ID (32-bit): A greater than 0 integer value shows the stream ID for which flow control is applicable, otherwise connection-level flow control.

Byte offset (64-bit): This unsigned integer notifies the maximum amount of data that can be sent on the open stream. However, in connection-level flow control stream cumulative data bytes from all open streams are to be considered.

4.3 Proposed ACK frame structure

A proposed ACK frame structure is shown in Figure 8. As per the working mechanism of a QUIC, there is a specific sequence of work-flow in which, window update state arises after the acknowledgement of the packet. In the execution of work-flow, an ACK and window update frames arrive separately one after another. The window update has been carried out based on the analysis of an ACK frame reception time. In a proposed scheme, the window update frame is coupled with the ACK frame instead of being sent separately. This refinement reduces the control signal overhead and the time to update size of the congestion window; this improves the packet transmission rate. In the proposed structure, a maximum size of the window update is 32-byte, which is fixed but can be made adaptive according to the network condition and application demand.

4.4 Proposed handshaking mechanism

A modified handshaking mechanism for the initial and repeated connection is shown in Figure 11.

Following are the steps to execute activities as per the proposed handshaking mechanism.

- In the prepared QUIC protocol based server-client setup, the initial or repeated connection request will take 1 or 0 RTT respectively.
- Sender receives an ACK after the successful reception of the data packet, whereas when receiver is disconnected sender stops receiving ACK. This may lead to an assumption that the receiver is temporarily disconnected. This controls the rate of transmission and freezes timers by sending back-off persist packet to the receiver till it receives an ACK. Once the congestion is under control, it restarts the frozen timers and the sender can transmit packets with the full rate.
- Lost data is resend by the sender with an updated window size and new ACK.
- If packet loss rate gradually increases, the receiver sends NACKs to the sender with a zero window size update, till congestion is under control. Once congestion is under control, the receiver updates the window size to the sender to resend the lost data.

5 EXPERIMENTAL SETUP

ModQUIC performance is verified for the following evaluation metrics and testbed environment.

5.1 Evaluation metrics

A ModQUIC performance is evaluated for throughput, delay, speed, and fairness.

**Throughput**: The amount of data in megabits that can be transferred by the network from a sender to a receiver during a time in seconds. A throughput is expressed in kilo or mega bits per second, which is given in Equation (14). In case of multiple flows, a throughput is the sum of throughput of all the flows.

\[
\text{Throughput} = \frac{\text{cwnd}}{\text{RTT}}. \tag{14}
\]

**Delay**: A delay in the network states how long it takes for a bit of data to travel across the network from one communication endpoint to another. The average delay given in Equation (15) is the sum of transmission delay (TD), propagation delay (PD), processing delay (PrD), and queuing delay (QD), expressed in milliseconds (ms).

\[
\text{Delay} = \text{TD} + \text{PD} + \text{PrD} + \text{QD}. \tag{15}
\]

Here, TD (transmission delay) is the amount of time required to push all the packet bits into the link, which is a function of the packet's length and data rate of the link. PD (propagation delay) is the amount of time required for a message to travel from the sender to the receiver, which is a function of distance over the speed with which the signal propagates. This is the function
of physical distance between the two entities and the speed of light. PrD (processing delay) is the amount of time required to process the packet header, check for bit-level errors and to determine the packet’s destination. QD (queuing delay) is the amount of time the packet is waiting in the queue until it can be processed.

**Fairness**: Fairness is represented in terms of the fairness index, which varies between 0 and 1. Fairness index is a measure when two or more applications compete to acquire network resources such as bandwidth, throughput, buffer space; when all are sharing equal resources, fairness index is 1. One of the popular methods used to measure fairness index is Jain’s fairness index given in [36] and expressed as in Equation (16).

Jain’s fairness index is a foremost fairness measure for TCP flows. This fairness index ranges from $1/n$ to 1. The value of this index tends to be 1 only if each flow has an equal share, and tends to be $1/n$ if a single flow acquires all network resources [7, 37].

$$F(f_1, f_2, \ldots, f_n) = \frac{\left(\sum_{i=1}^{n} f_i\right)^2}{n \cdot \sum_{i=1}^{n} f_i^2}, \quad (16)$$

where $n =$ number of flows sharing the resource, $f_i =$ resource allocation for $i^{th}$ flow.

### 5.2 Parameter space used and testbed environment

The testbed environment and assembled setup is shown in Figures 12 and 14, which are utilized to carry out the experimental evaluation of the ModQUIC, QUIC, and TCP performance. However, Figures 13 and 15 are used to carry out the evaluation of ModQUIC and TCP/HTTP2 for a browser network.

A testbed, shown in Figure 14 has been set up using the OpenFlow Mininet platform, which creates the network of virtual hosts. A Mininet host runs standard Linux network software and supports OpenFlow for a highly flexible custom routing and the software-defined networking. The libquic library with a ‘golang’ programming language created by Google is used as a platform package to analyze the performance. The output results are obtained by the libquic analysis script written in the Python programming. The QUIC toy-server-client program is used to analyze the performance. The client tries to establish a connection with the server for an FTP via the external host machine. To perform an experimentation, hosts are at a one hop distance away from the server. The performance is observed by changing the loss rate and link bandwidth using wondershaper, a traffic shaping tool.

The testbed shown in Figure 15 has been set up with a dummy QUIC server-client model available in the Chromium browser code-base (https://code.google.com/p/chromium/). For the experimentation, a TCP server application with TCP-CUBIC functionality is used. To add the ModQUIC functionality and to capture the relevant variables, QUIC source code is modified. On the client-side, two different configurations of the Chromium, ModQUIC, and TCP are enabled (QUIC disabled) and deployed. This implementation of a ModQUIC server is used to test results from a MacBook Pro laptop (running chromium-browser built from source) which has been used to carry out measurements3. It has been noted that implementation is meant for the integration-testing and not to test performance at the scale.

3 Intel Core i5 CPU @2.5GHz with 4GB RAM running MacOS Sierra 10.12.6 (64-bit). With chromium version of 63.0.3229.0 (64-bit) and Darwin kernel version 16.7.0.
5.2.1 System Configuration

- Server-client configuration: Intel Core\textsuperscript{TM} i5-2400 CPU @ 3.10 GHz X4, 8 GB RAM, Ubuntu 14.04 LTS, 64-bit operating system.
- Hop configuration: Intel Core\textsuperscript{TM} i5 CPU M560@2.67 GHz X4, 4 GB RAM, Ubuntu 14.04 LTS, 64-bit operating system.
- Internet connectivity: 1.9 Gbps across 4 lease lines with (2:1:1) load balancing with core switch 2X VDX8770-8, 1X VDX6740-T and router configuration of 2 X 2 MIMO, 433 Mbps/client on 5 GHz, 72 Mbps/client on 2.4 GHz.

5.2.2 Analysis and traffic shaping tools

A iPerf tool running on the client machine is used to measure link bandwidth available between the server and client. A wondershaper tool is used to manage traffic, to fix packet loss and to allow the propagation delay to be set on a client machine. It can also be used for setting up the packet loss rate while downloading the data in the browser to create a loss effect.

6 RESULT ANALYSIS

The comparative performance analysis of the ModQUIC, QUIC and TCP have been carried out for the throughput and delay. This performance is calculated based on the average datarate achieved and the packet sent per RTT, respectively. However, the fairness analysis is used to verify the fair resource allocation.

6.1 Throughput and delay analysis

Figures 16, 17, and 18 show the performance comparison between ModQUIC, QUIC, and TCP concerning the loss rate for a different link bandwidths. This has been accomplished based on the achieved average data rate using generated ACKs. At a loss of 0% and when the sufficient link bandwidth is available, TCP outperforms due to its initial aggressiveness, whereas the performance of ModQUIC and QUIC is almost similar. The TCP performance gradually decreases at the rate of 0.21Mbps per percent of the loss rate. In ModQUIC and QUIC, due to multiplexed streams and an out of order delivery, better performance compared to TCP has been observed in the lossy link. However, ModQUIC is better than QUIC due to bandwidth occupancy limitation which in turn depends on the used window size. Even though the slow start is avoided in the QUIC, its default window size is updated only for an analysis of a previously sent packet's success rate and rate of transmission. In the ModQUIC, maximum bandwidth utilization is observed, which in turn is responsible for the window update. This fine-tuned window update mechanism per ACK reception would result in the reception of more packets within a specified time and hence, as a whole ModQUIC outperforms.

For a lossless bottleneck link of 5 Mbps and 10 Mbps, all the three flows were closely competing with each other. As loss rate increases, the link becomes congestive, a reaction to this TCP reduces data rate gradually. However, it has been observed that the performance of a ModQUIC is improved by 21% over a TCP and 3.43% over the QUIC, due to built on the top of UDP and fine-tuned window update mechanism. For sufficient bandwidth (50 Mbps) and lossless link, TCP dominates ModQUIC and QUIC due to packet pacing creating an overhead. Once the loss rate is increased, the performance of TCP suddenly drops down below ModQUIC and QUIC.

The delay is measured in terms of time required to send packets per RTT for a different link bandwidth and loss rate (e.g.
TCP0, QUIC0, and ModQUIC0 indicates performance of TCP, QUIC, and ModQUIC for 0% loss, a similar representation is considered for 1% and 5% as shown in Figures 19–21. Experimental results shown in Figures 22 and 23 further investigates the cwnd growth with respect to time. To extract cwnd variation, a ModQUIC and QUIC source code is equipped, whereas tcpprobe [38, 39] is used for TCP. Figures 22 and 23 show cwnd growth for link bandwidths of 5 Mbps and 50 Mbps with a 2% loss. This shows a smooth cwnd variation as well as maximum bandwidth utilization in ModQUIC compared to QUIC and TCP. When competing with TCP and QUIC, ModQUIC can achieve a larger bandwidth share. It is observed that while all protocols use cubic congestion control scheme, Mod-

QUIC and QUIC increases its window size more aggressively (both in terms of slope, and in terms of more frequent window size increases). As a result, ModQUIC and QUIC can grab available bandwidth faster than a TCP does, leaving TCP unable to acquire its fair share of the bandwidth. This unfairness is observed at the early stage of the data transfer, but fairness is improved with time and file size. Table 3 shows the ModQUIC performance over TCP/HTTP2 in terms of throughput and speedup for single and multiple flows. The throughput characteristics drastically vary when ModQUIC and TCP are competing for flows as opposed to none at all. With extensive experimentation in a live network; results show that ModQUIC throughput outperforms TCP/HTTP2, especially for single dominant flow. In case of multiple streams (flows), two from each of the ModQUIC and TCP/HTTP2, HTTP2 creates multiple dedicated connections to serve each flow. However, the ModQUIC uses multiplexed UDP streams in addition to dedicated connections which multiplexes TCP streams. This causes a fall in the TCP packet transmission rate that results in the throughput improvement of ModQUIC over TCP.

The result analysis is presented after 50 iterations (replications) for a file size of 10–50 MB and 5–50 Mbps bandwidth by adding upto 10% of loss with RTT 20 ms and 50 ms. These results are verified with the referred four step verification process (Tables 2 and 3) with confidence interval greater than 90%.
TABLE 1 Parameter space used for experimentation

| Category                  | Parameters      | Values                                      |
|---------------------------|-----------------|---------------------------------------------|
| Network parameters        | Bandwidth       | 5 Mbps, 10 Mbps, 50 Mbps                    |
|                           | Link capacity   | 72 Mbps/client for 2.4 GHz and 433 Mbps/client for 5 GHz |
|                           | RTT             | 20 ms, 50 ms                               |
|                           | Packet loss     | 0%, 2%, 5%, 10%                            |
| Server side               | Video file size | 10 MB, 30 MB, 50 MB                        |
|                           | Clients connected | 3 (with virtual node) 2 (with browser)    |

TABLE 2 Throughput and speedup for ModQUIC against TCP/HTTP2 for various video file sizes

| No. of flows | Parameters                  | 10 MB | 30 MB | 50 MB |
|--------------|-----------------------------|-------|-------|-------|
| 1            | ModQUIC throughput (Mbps)   | 0.82  | 0.88  | 0.68  |
|              | TCP/HTTP2 throughput (Mbps) | 0.69  | 0.73  | 0.52  |
|              | Speedup (ModQUIC over TCP/HTTP2) | 1.18  | 1.21  | 1.31  |
| 4            | ModQUIC throughput (Mbps)   | 0.61  | 0.78  | 0.91  |
|              | TCP/HTTP2 throughput (Mbps) | 0.57  | 0.71  | 0.79  |
|              | Speedup (ModQUIC over TCP/HTTP2) | 1.07  | 1.09  | 1.16  |

6.2 Fairness analysis

ModQUIC, QUIC, and TCP flows are competing for bottleneck link of a 5 Mbps, 10 Mbps, and 50 Mbps for different RTT values and loss rates. The observations reveal that the ModQUIC is the fair solution to serve multiple streams sharing a bottleneck link.

The procedure has been carried out to test competing flows serviced by ModQUIC, QUIC, TCP, and TCP/HTTP2. ModQUIC, QUIC, and TCP flows were created with virtual nodes using the OpenFlow Mininet platform, whereas separate browsers are opted to create TCP/HTTP2 flows. The TCP/HTTP2 flows are generated using Mozilla Firefox and Opera. The multiplexing nature of TCP/HTTP2 causes the use of a single TCP/IP connection which limits testing durability under multiple flows. The video files of sizes 10 MB, 30 MB, and 50 MB were used and serviced by any or all the flows. Similarly, the same size files were used and serviced by TCP/HTTP2 using a browser network.

The available bandwidth and RTT were checked during each epoch with the Speedtest, a network monitoring tool [40], and each corresponds to the video file (e.g., 18 Mbps bandwidth and 54 ms RTT for the 1 MB file download). The parameter space used for experimental analysis is given in Table 2.

Figures 24 and 25 show the fairness performance calculated based on the Jain’s fairness index. In this fairness analysis graph, values considered are the average values of 5 Mbps, 10 Mbps and 50 Mbps links. A fairness index increases with the file size and mostly for long live traffic, it is very good. The requests service using TCP/IP acquired more bandwidth to serve smaller file sizes, which shows unfairness in bandwidth utilization.

7 Conclusion

QUIC is a predominant protocol and better choice as an alternative to TCP. ModQUIC is a transport and application layer solution, which enhances the throughput, reduces the latency and is easily deployable in an existing network. To determine the rate of transmission a death-birth process queuing model is used and the window update information is the function of steady-state probability. The performance of the proposed modification is verified by using two different testing environments. The result analysis is carried out by using the throughput, delay in the presence of loss for limited and sufficient bandwidth. A network throughput using ModQUIC is improved by 35% and 51.93% over QUIC and TCP, respectively, whereas a marginal reduction in the delay is observed. The performance of TCP for a lossy link is very poor, whereas ModQUIC and QUIC are found to be a better and stable. It has been observed that in case of sufficient bandwidth, the QUIC and ModQUIC protocols creates bottleneck. It has been observed that the throughput of a network with the ModQUIC and QUIC is better for the high BDP and large file sizes. The fairness analysis shows that the fairness index improves with respect to the file
size and BDP. In this experimental analysis, it is observed that the limitations of the ModQUIC are mainly due to cubic functionality and to overcome high BDP performance limitations, alternative congestion control mechanism to CUBIC which is more aggressive in slow start phase (NewReno) or adaptive with respect to bottleneck bandwidth like BBR may be useful solutions.

REFERENCES

1. VNI Cisco, Cisco Visual Networking Index: Forecast and Methodology 2016–2021. Cisco (2017)
2. Elkhahth, Y., et al.: Can SPDY really make the web faster? In: IEEE Networking Conference (IFIP), pp. 1–9. IEEE, Piscataway (2014)
3. Wei, S., Swaminathan, V.: Low latency live video streaming over HTTP 2.0. In: Proceedings of Network and Operating System Support on Digital Audio and Video Workshop. ACM, New York (2014)
4. Roskind, J.: QUIC: Design Document and Specification Rational (2013)
5. Claeyts, M., et al.: Deadline-aware TCP congestion control for video streaming services. In: 12th IEEE International Conference on Network and Service Management (CNSM), pp. 100–108. IEEE, Piscataway (2016)
6. Dong, M., et al.: PCC: Re-Architecting Congestion Control for Consistent High Performance, vol. 1.2. NSDI (2015)
7. Hassan, M., Jain, R.: High Performance TCP/IP Networking, vol. 29. Prentice Hall, Englewood Cliffs (2003)
8. Wang, S., et al.: How speedy is SPDY?, vol. 1.1. NSDI (2014)
9. Vernersson, A.: Analysis of UDP-based Reliable Transport using Network Emulation. (2015)
10. Kharat, P., Kulkarni, M.: Modified QUIC protocol for improved network performance and comparison with QUIC and TCP. Int. J. Internet Protocol Technol. 12(1), 35–43 (2019)
11. Roskind, J.: QUIC: Design Document and Specification Rational (2015)
12. Ha, S., et al.: CUBIC: A new TCP friendly high-speed TCP variant. ACM SIGOPS Operat. Syst. Rev. 42, 64–74 (2008)
13. Xu, L., et al.: Binary increase congestion control (BIC) for long-distance networks. In: INFOCOM 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 4, pp. 2514–2524. IEEE, Piscataway (2004)
14. Cardwell, N., et al.: BBR Congestion Control. Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-icetcb-bbr-congestion-control-00 (2017)
15. Leith, D., Shorten, R.: H-TCP: TCP for High-speed and Long-distance Networks. In: Proceedings of PFLDnet (2004)
16. Langley, A.: Wan-Tech:Chan gro: Quic Crypto (2013)
17. Sveet, L.: QUIC: Congestion Control Land Loss Recovery (2015)
18. Sveet, L., Iyengar, J.: QUIC Loss Recovery and Congestion Control. Internet-Draft, Intended status: Informational, Internet Engineering Task Force (2015)
19. Iyengar, J.: QUIC: Redefining Internet Transport (2016)
20. Gizis, A.: Taking Google’s QUIC for a Test Drive (2016)
21. Hamilton, R., et al.: QUIC: A UDP-based Secure and Reliable Transport for HTTP/2. IETF, draft-tsvwg-quic-protocol-02 (2016)
22. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. 2017, draft-ietf-quic-transport-01 (work in progress)
23. Chromiumblog: ‘A QUIC update on Google’s experimenta l transport. http://blog.chromium.org/2015/04/quick-update-on-googles-experimental.html. Accessed May 25, 2017
24. Langley, A., et al.: The QUIC transport protocol: Design and internet scale deployment. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 183–196. ACM, New York (2017)
25. Carlucci, G., et al.: HTTP over UDP: An experimental investigation of QUIC. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 609–614. ACM, New York (2015)
26. Megyesi, P., et al.: How quick is QUIC? In: International Conference on Communications (ICC), pp. 1–6. IEEE, Piscataway (2016)
27. Biswal, P., Gnawali, O.: Does QUIC make the web faster? Global Communications Conference (GLOBECOM), pp. 1–6. IEEE, Piscataway (2016)
28. Das, S.R.: Evaluation of QUIC on web page performance. Ph.D. Dissertation, Massachusetts Institute of Technology (2014).
29. Cook, S., et al.: QUIC: Better for what and for whom? In: IEEE International Conference on Communication (ICC), pp. 1–6. IEEE, Piscataway (2017)
30. Srivastava, A.: Performance analysis of QUIC protocol under network congestion. Ph.D. Dissertation, Worcester Polytechnic Institute (2017)
31. Kakki, A. M., et al.: Taking a long look at QUIC. In: Proceedings of the 2017 Internet Measurement Conference. ACM, New York (2017)
32. Gallo, Y.D.M., et al.: Towards a scalable modular QUIC server. In: Proceedings of the Workshop on Kernel-Bypass Networks, pp. 19–24. ACM, New York (2017)
33. Coninck, Q.D., Bonaventure, O.: Multipath QUIC: Design and evaluation. In: Proceedings of the 13th International Conference on emerging Networking Experiments and Technologies, pp. 160–166. ACM, New York (2017)
34. Hussein, A., et al.: SDN for QUIC: An enhanced architecture with improved connection establishment. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, pp. 2136–2139. ACM, New York (2018)
35. Shelton, M., Ross, S.M.: Stochastic Processes, vol. 29. Wiley, New York (1996)
36. Jain, R., et al.: A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System, vol. 38. Eastern Research Laboratory, Digital Equipment Corporation, Hudson (1984)
37. Kharat, P., et al.: QUIC protocol performance in wireless networks. In: International Conference on Communication and Signal Processing (ICSP), IEEE, Piscataway (2008)
38. TCP probe. https://wiki.linuxfoundation.org/networking/tcpprobe. Accessed July 18, 2018
39. Persson, A., et al.: TCP Probe: A TCP with built-in path capacity estimation. In: Proceedings of the 8th IEEE Global Internet Symposium. IEEE, Piscataway (2016)
40. Speedtest networking monitoring tool. http://beta.speedtest.net/. Accessed July 18, 2018

How to cite this article: Kharat P, Kulkarni M. Modified QUIC protocol with congestion control for improved network performance. IET Commun. 2021;15:1210–1222. https://doi.org/10.1049/cmu2.12154