Modified QUIC protocol for improved network performance and comparison with QUIC and TCP

Prashant Kharat* and Muralidhar Kulkarni

Department of Electronics and Communication,
National Institute of Technology Karnataka,
Surathkal, Mangalore, India
Email: prashant.kharat@walchandsangli.ac.in
Email: mkulkarni@nitk.ac.in
*Corresponding author

Abstract: Congestion control mechanism is solely responsible for maintaining the performance of streaming data. However, when there is no congestion, a regular delivery window update is followed as a step by step process. The process can be improved by individual window update along with acknowledgement (ACK) as feedback to the server even in the absence of congestion. To achieve maximum throughput and minimum delay, we have suggested modification in the existing handshaking mechanism of quick UDP internet connections (QUIC) protocol. This paper presents an investigation of QUIC protocol performance and proposes a modified QUIC (ModQUIC): a modification in existing handshaking mechanism of QUIC to reduce control overhead. Chromium server-client model testbed setup results, show that the proposed technique gives stable output and improves performance in terms of overall throughput and delay over QUIC and transmission control protocol (TCP). Performance has been tested for limited (2 Mbps) and sufficient (10 Mbps) link bandwidth in presence of loss. The validation of results has been carried out with the help of linear regression model. The result show a throughput improvement of 35.66% and 51.93% over QUIC and TCP respectively and also the delay is reduced by 3% to 5% over QUIC and TCP.

Keywords: modified QUIC; ModQUIC; quick UDP internet connections; QUIC; network performance; congestion control.

Reference to this paper should be made as follows: Kharat, P. and Kulkarni, M. (2019) 'Modified QUIC protocol for improved network performance and comparison with QUIC and TCP', *Int. J. Internet Protocol Technology*, Vol. 12, No. 1, pp.35–43.

Biographical notes: Prashant Kharat is pursuing his PhD in Electronics and Communication Engineering in the Department of Electronics and Communication Engineering at National Institute of Technology Karnataka, Surathkal. He received his BE and ME in Electronics Engineering with specialisation in Computer Science from the Shivaji University, Maharashtra, India in 2001 and 2007. His current research interest include wireless networks, congestion control in data networks, internet technology and cooperative communication. He has published and presented papers in the subjects cooperative communication, wired-wireless networks and soft computing techniques and related applications in journals and conferences.

Muralidhar Kulkarni is a Professor in the Department of Electronics and Communication Engineering, National Institute of Technology Karnataka, Surathkal. He obtained his BE in Electronics Engineering from the Bangalore University, Karnataka, India, MTech in Satellite Communication and Remote Sensing from the IIT, Kharagpur, India and PhD from the JMI Central University, New Delhi, India. He has published several research papers in national and international journals/conferences of repute. He is also the author of popular text books in microwave and radar engineering, communication systems, digital communications and information theory and coding. His teaching and research interests are in the areas of digital communications, fuzzy digital image processing, optical communication and networks and computer communication networks.

1 Introduction

Today's fast paced world demands data transfer with almost zero latency. Online business is an example wherein if a vendor's website page load time (PLT) is high; results in poor customer satisfaction. Internet protocol (TCP/IP) is the most popularly used protocol to access internet data using wired or wireless technology. To maximise the customer satisfaction in traditional information gathering or online services; it is necessary to monitor TCP/IP network parameters to improve performance (Grigorik, 2017; Hassan and Jain, 2003). TCP/IP has contributed extensively to the

networking industry by delivering remarkable results. Google has a detailed analysis of TCP/IP performance as 30% to 35% of all internet traffic passes through its servers. In addition to this, Google's Chrome browser is the most popular browser with approximately 40% of market share (Flach et al., 2016). Based on the experience, in 2013, Google developed a new protocol using user datagram protocol (UDP) called as quick UDP internet connections protocol. QUIC protocol is a simple and on top of UDP, suitable for supporting application protocols for booting (Figure 1).

In this work, comprehensive goal is to investigate and understand the benefits and tradeoffs of QUIC protocol. QUIC protocol development and deployment is a challenging project by Google, basically developed to improve the network throughput and reduce latency. QUIC protocol source code is publicly available which can be improved further and test the results as well. Hence, while understanding the QUIC basic structure and working model, we realise an idea to improve throughput and reduce latency by modification in handshaking mechanism. Following are the key challenges:

- Within past few years, due to growth rate of Web technology, there is a demand in higher speed of operation and faster data rate requested by the users. Higher speed leads to faster and better user satisfaction, which results in user retention.
- QUIC protocol has been developed to compete with TCP which is the most dominant and worldwide deployed transport layer protocol.
- QUIC protocol source code is publicly available of maybe possible that there is a gap between publicly available and actually deployed on Google client.
- QUIC protocol is under rapid development since 2013; 43! stable versions are released. Limited literature and experimental studies are available which become obsolete before publication.

To improve overall throughput and reduce latency, this modification proposes transport layer solution using QUIC as a base protocol. There are different opinions about QUIC location in protocol stack whether it should be at transport or application layer. This work investigates QUIC performance and proposes a modified handshaking mechanism to reduce control overhead. A testbed has been set up with Chromium server-client model and traffic shaping tool. Results show that performance of QUIC is better than TCP and propose modification in QUIC substantially improves throughput with a marginal reduction in delay and stable with respect to loss rate.

Rest part of the paper has four sections Section 2 gives insight into background information and related experimental work of QUIC protocol with features and congestion control mechanism. In Section 3 detail discussion of the proposed work is presented. Section 4 describe experimentation part carried out to investigate proposed work and result analysis. Finally, work is summarised and concluded in Section 5.

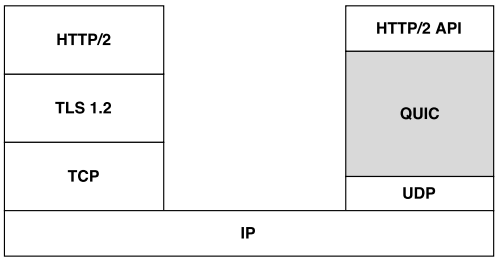
2 QUIC protocol background with congestion control and related work

This section provide background of QUIC protocol and survey on experimental work by various researchers related to this work.

2.1 Background

QUIC is an experimental transport layer protocol proposed by Roskind (2012). It is a secure and multiplexed protocol with UDP as a base protocol. Using bandwidth estimation technique it supports flow and congestion control with reduced connection and transport latency (Hamilton et al., 2016).

Figure 1 QUIC placement in protocol stack



2.1.1 QUIC features

QUIC has following highlighted features:

- Multiplexed steams over UDP connection allows out of order delivery solving head of line (HoL) blocking.
- Less connection establishment latency using reduced handshaking mechanism.
- Packets authentication and payload encryption for secure data delivery.
- Connection and stream level flow control mechanism.
- Flexible and pluggable congestion control mechanism.
- 64 bit connection ID (randomly generated by client) for connection migration.
- Packet pacing mechanism for handling bursty data.

2.1.2 QUIC's congestion control mechanism

Various congestion control mechanisms suggested by researchers (Afanasyev et al., 2010; Luckie et al., 2014), out of those QUIC uses CUBIC with packet-pacing mechanism. At the beginning of the connection, end-hosts can negotiate the technique to employ for congestion control. Ha et al. (2008) proposed an enhanced version of binary increase congestion-control (BIC) (Xu et al., 2004) as a CUBIC, in which round trip time (RTT) independent congestion window growth function is used. To achieve this CUBIC picked up H-TCP (Leith and Shorten, 2004) approach to calculate $cwnd$

size [equation (1)], which is cubic function of elapsed time t since last congestion event.

$$W_{CUBIC} = C(t - (\frac{\beta.W_{max}}{C})^{\frac{1}{3}})^3 + W_{max} \quad (1)$$

where

W_{CUBIC} is *cwnd* size

W_{max} is a *cwnd* just before last window reduction

C is predefined constant (scaling factor)

β is decrease factor.

Equation (1) shows that it preserves BIC properties, RTT fairness, limited slow start and rapid convergence. As an additional precaution, CUBIC uses mechanism to ensure performance to be equal or better than standard Reno with simultaneous checking and computation of W_{reno} parameter. With experimental studies, it has been seen that performance and fairness property of CUBIC are acceptable. TCP-CUBIC is available in Linux TCP suite (Kernel Version 2.6.16) and is currently the most widely used congestion control mechanism. To handle bursty data, packet-pacing mechanism is used in which inter-arrival and inter-departure time of two consecutive packets are adjusted. In packet-pacing, sending data rate is calculated based on the relative forward delay, which is the difference between inter-arrival times at receiving and sending end for same consecutive data packets (Aggarwal et al., 2000).

2.2 QUIC related experimental work

Most of the literature available for QUIC is in the form of internet drafts proposed by researchers from Google. Loss recovery and congestion control (Swett and Iyengar, 2015; Swett, 2015), secure QUIC-crypto connection (Langley and Chang, 2013), redefined QUIC (Iyengar, 2016), QUIC used for test drive (Gizis, 2016), enhanced draft of QUIC (Iyengar and Thomson, 2017), are different internet drafts of QUIC protocol.

Researchers so far have analysed the performance of QUIC with web PLT. Megyesi et al. (2016) perform network tests for high bandwidth, high packet loss and large and small objects on a web page. Carlucci et al. (2015) tested QUIC performance with respect to goodput, bandwidth utilisation and PLT. They investigated that forward error correction (FEC) mechanism reduces QUIC performance. Das (2014) in his MS work analysed QUIC performance with more than 500 web pages and compared with TCP over limited bandwidth and high RTT. He noticed that QUIC outperforms for small size web pages with fewer objects. Biswal and Gnawali (2016) reported that QUIC outperforms hyper text transfer protocol (TCP/HTTP) with respect to PLT in the presence of loss for large object size. Cook et al. (2017) used local and remote testbeds to identify where the QUIC protocol is most efficient. They have investigated QUIC performance with respect to the type of access network and with packet loss and delay generation in the link. They concluded that QUIC outperforms HTTP/2 over transport layer security (TCP/TLS) in unstable networks such as wireless/mobile but in case of stable and reliable networks there has been no significant contribution. Srivastava (2017) in his MSc thesis has been compared QUIC performance with TCP with respect to throughput, delay and fairness. He has found that for an added delay and loss QUIC outperform and in case of competing flow QUIC acquire more bandwidth share than TCP. Kakhki et al. (2017) carried out extensive experimentation for a variety of network conditions. They found that QUIC outperforms TCP/HTTP in nearly every scenario, QUIC is very sensitive with out of order delivery and shows very poor performance. They found that QUIC is unfair when competing with TCP flows.

Table 1 Comparative contribution analysis

Contributer	*QUIC version used*	*Performance analysis*						*Testing environment*	*New research contribution*
		Throughput (goodput)	*Loss*	*Delay*	*Fairness*	*PLT*	*Link bandwidth*		
Megyesi et al. (2016)	20	N	Y	N	Y	Y	N	Wire	N
Carlucci et al. (2015)	21	Y	Y	N	N	Y	Y	Wire	N
Das (2014)	23	N	N	N	N	Y	N	Wire	N
Biswal and Gnawali (2016)	23	Y	N	N	N	Y	Y	Wire	N
Cook et al. (2017)	25	N	Y	Y	N	Y	N	Wire/ wireless	N
Srivastava (2017)	25 to 36	Y	Y	Y	Y	N	Y	Wire	N
Kakhki et al. (2017)	25 to 37	N	Y	N	Y	Y	Y	Wire/ cellular	N
Our work	*25 to 39*	*Y*	*Y*	*Y*	*N*	*N*	*Y*	*Wire*	*Y*

It may be seen that, earlier researchers only carried out an experimental investigation of QUIC with released versions of the protocol (summarised in Table 1). This work has added contribution in terms of modifying existing working mechanism of QUIC compared to previous contributors (detail discussion is given in Section 3).

3 Proposed work

Existing handshaking mechanism is basically related to rate control. Acknowledgement (ACK) frame and window update frame are mainly used for rate control in QUIC. In QUIC protocol working model, window update state comes after reception of ACK frame, which depends on ACK frame reception time analysis. In ACK frame reception time analysis, if current ACK reception time is less than previous ACK frame reception time then only window updated to next value by a factor α.

This paper proposes a change in the above stage of the working model of QUIC. In the proposed model, window update frame is attached with ACK frame, instead of being sent separately. This change, updates window size with every ACK frame reception, which reduces one transition state of the protocol. This results in a reduction of state transition delay. The suggested modification window size, varies with every ACK reception, that results in smooth variation in congestion window size. This smooth congestion window variation regulates network traffic, which helps congestion control.

Following subsections provide an overview of ACK frame and window update frame structure whereas in detail description about the proposed frame structure.

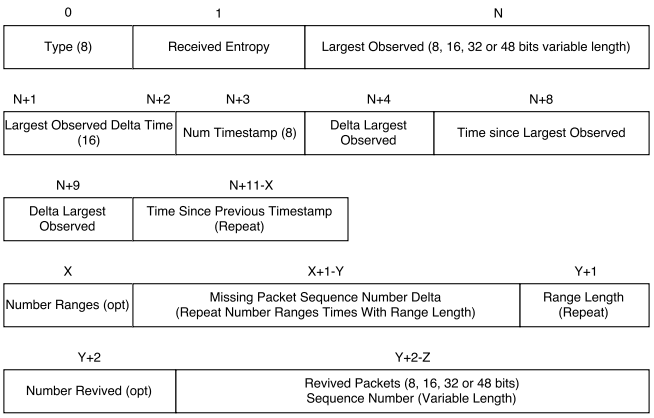
3.1 ACK frame structure

The ACK frame structure as shown in Figure 2, is used by the receiver to inform server about received and missing packets (Hamilton et al., 2016). Structure of QUIC-ACK frame is different from TCP-ACK frame. In QUIC, not an ACK (NACK) indicates gaps in received packets. The server periodically sends stop-waiting frame to inform the receiver to stop waiting for packets below a particular sequence number. This leads to higher number of least unacked packets at the receiver.

3.2 Window update frame structure

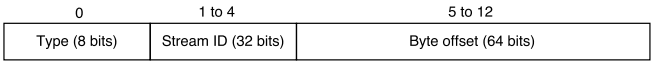
The window update frame is used to inform the peer of an increase in end point's flow control receive window (Hamilton et al., 2016). In the frame structure shown in Figure 3, stream ID field is zero correspond to connection-level flow control and greater than zero imply that flow control is applicable for that specific stream. Violating flow control by sending more bytes than prescribed will result in receiving endpoint closing the connection.

Figure 2 ACK frame structure



Source: Hamilton et al. (2016)

Figure 3 Window update frame structure



Source: Hamilton et al. (2016)

The default window size is 16 KB both at stream and connection level and continuously increases during handshaking by exchanging window control parameters.

The fields in the window update frame are as follows:

- Frame type: this must be set to 0×004 to indicate this as a window update frame.
- Stream ID: integer value greater than zero indicates the stream ID whose flow control window is being updated, otherwise connection-level flow control.
- Byte offset: unsigned integer used to send maximum data on the open stream and for connection-level flow control, cumulative data bytes from all open streams will be considered.

QUIC has the window update functionality managed with window update frame. Relevance to window update mechanism uses a queuing model, based on pure birth-death process (Ross, 1996). For congestion state estimation, packet emission probability based on previous state is considered. To control packet transmission rate, three state window update strategy given in equations (2) to (4) are used at the source end.

Let the window update information be W_u:

$$W_u = 0, \text{ if } P_n = P_{n-1} \tag{2}$$

$$W_u = 1, \text{ if } P_n > P_{n-1} \tag{3}$$

$$W_u = -1, \text{ if } P_n < P_{n-1} \tag{4}$$

where

n number of packets in the network

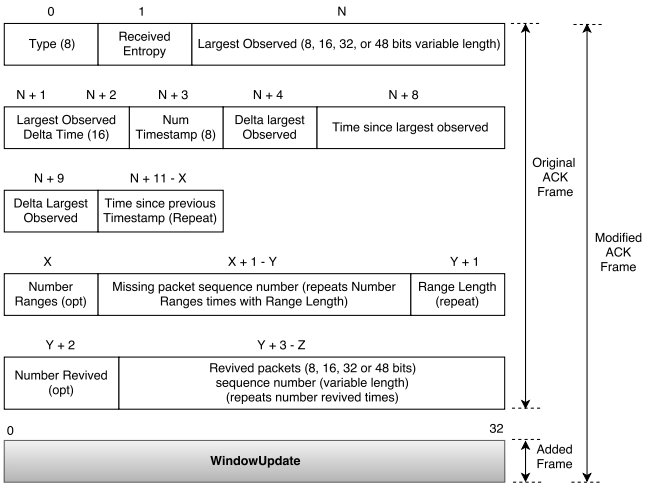
P_n steady state probability of n packets

P_{n-1} steady state probability of $n-1$ packets.

Under steady state condition $n > 0$, whereas, P_n is a function of λ_n (arrival rate) and μ_n (departure rate). Following are the steps to control congestion and time out events.

- If size of the packet decreases, number of packets in the network get increased, to send the same size of data. This will result into increment in probability of collision and hence decrement in window size leading to control in congestion.
- The packet size is dependent on the available bandwidth (allowed rate to send the data) which ultimately depends on the window size.
- The window update information will contain state to decrease the window size and to control the rate if congestion is detected.
- As flow control is achieved, simultaneously packet size can get decreased which will lead to an increment in a number of packets as per state 1. To keep less probability of packets emitted the rate has to be controlled.
- The outcome of this approach is to control the congestion which results an improvement in system performance.

Figure 4 Proposed ACK frame structure



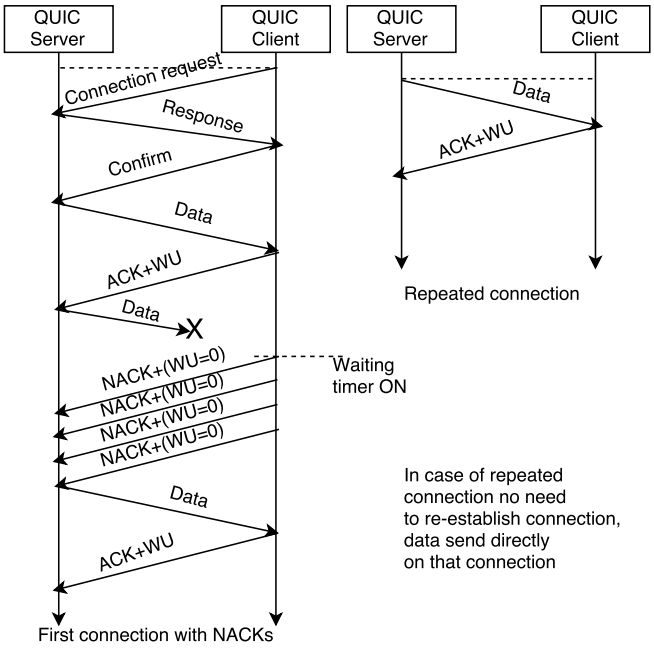
3.3 Proposed ACK frame structure

A proposed frame structure is marked as a modified ACK frame in Figure 4. With respect to working structure of QUIC, window update state is followed by packet ACK state. In the proposed modification, window update frame is attached with ACK frame to reduce control overhead and window update delay. This results into smooth variation in $cwnd$ and improvement in throughput. The window update size varies according to network condition and application demand.

3.4 Proposed handshaking mechanism flow diagram

In line with the proposed frame structure, a modified handshaking mechanism for first and repeated connection shown in Figure 5.

Figure 5 Proposed handshaking mechanism



The following steps are the flow for proposed handshaking mechanism.

- After creating QUIC-based server-client model, first connection establishment process is carried out. It takes zero or 1-RTT based on initial or repeated connection request.
- On receiving a data packet, the receiver sends ACK to the sender. In case the receiver is disconnected, the sender stops receiving the ACK and assumes that the receiver has been temporarily disconnected. Then sender controls rate of transmission and freezes timers by sending back-off persist packet to the receiver till it receives ACK. As soon as the congestion is under control, it restarts frozen timers. This results in transmission with full rate by the sender.
- If data sent is lost, the sender re-sends data and updates window size by sending ACK + window update.
- If the same situation is repeated, the receiver sends ACK (with NACK) to the sender continuously with zero window update until congestion is under control. Once congestion is under control, receiver updates the window size to sender and resends the lost data to receiver.

4 Experimentation

4.1 Evalution metrics used

Performance evaluation of ModQUIC, QUIC and TCP is carried out with respect to overall throughput and delay for different bandwidth and loss rate.

4.1.1 Throughput

The amount of data that can be transferred by the network from a sender to a receiver during a period of time expressed in Kilo or Mega bits per second (Kbps or Mbps). In case of multiple flows, throughput is the sum of throughput of all flows.

4.1.2 Delay

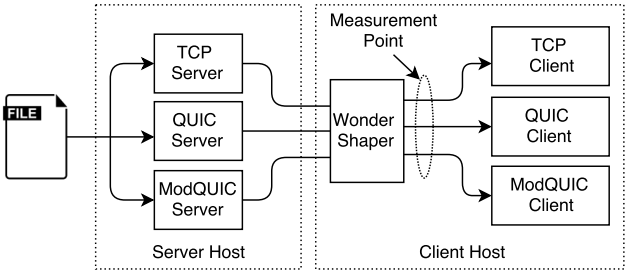
A period of time required for a packet to reach receiving end. Normally in networks total delay is the sum of transmission delay, propagation delay, processing delay and queuing delay, expressed in mili-seconds (ms).

4.2 Experimental setup

4.2.1 Server-client configuration

Performance testing has been done through a dummy QUIC server-client model present in the Chromium browser code-base available at https://code.google.com/p/chromium/. Figure 6 shows testbed environment with QUIC version 33 (actually tested for QUIC versions 25 to 39, but as there were not significant change in result, presented results for version 33). The data generation is performed by using Google certified www.example.org. For this investigation TCP server application with TCP-CUBIC functionality is used. QUIC source code is modified to add ModQUIC functionality and logged relevant variables. On client side three different configurations ModQUIC enabled, QUIC enabled and TCP enabled (QUIC disabled) of Chromium are deployed. Experimentation is carried out multiple times by using loopback technique to increase the traffic and create logs of almost ten thousand packets with payload size of 270 bytes.

Figure 6 Testbed environment



4.2.2 System configuration

Intel® Core^{TM} i5-2400 CPU @ 3.10 GHz × 4, 8 GB RAM, Ubuntu 14.04 LTS, 64-bit operating system. Chrome version 63.0.3239.132. Linux Kernel Version 4.4.0-93-generic.

4.2.3 Internet connectivity

1.9 Gbps across four lease lines with (2:1:1) load balancing with core switch 2xVDX8770-8, 1xVDX6740-T and router configuration of 2×2 MIMO, 433 Mbps/client on 5 GHz, 72 Mbps/client on 2.4 GHz.

4.2.4 Analysis and traffic shaping tools

iPerf: a network performance measurement tool is run on the client machine to measure an amount of data transferred and bandwidth available for server-client. Whereas wondershaper tool deployed on the client is used for managing traffic, to manipulate bandwidth, to fix packet loss and allow propagation delay to be set.

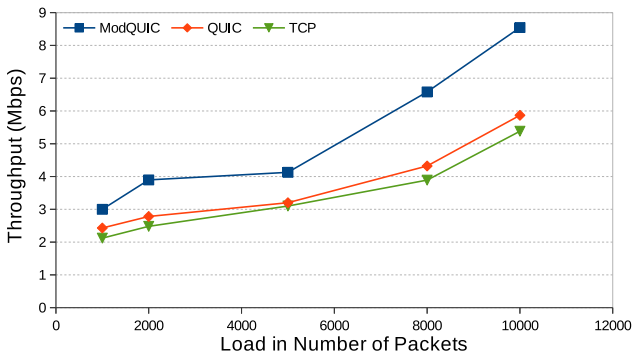
4.3 Experimental results

Experimental results shows comparative analysis of ModQUIC, QUIC and TCP performance. In this experiment base RTT value of 20 ms and queue size is equal to bandwidth delay product (BDP) with drop-tail approach are considered.

Table 2 Throughput performance comparison

No. of packets	*Throughput (Mbps)*			*% improvement of ModQUIC over QUIC*
	ModQUIC	*QUIC*	*TCP*	
1,000	2.999	2.430	2.120	23.41
2,000	3.896	2.781	2.480	40.09
5,000	4.128	3.201	3.100	29.51
8,000	6.581	4.322	3.890	52.26
10,000	8.544	5.867	5.382	50.24

Figure 7 Throughput performance comparison (see online version for colours)



In QUIC window update is sent from the client only, when there is congestion based on analysis of ACK reception time. Whereas in ModQUIC, every ACK is associated with

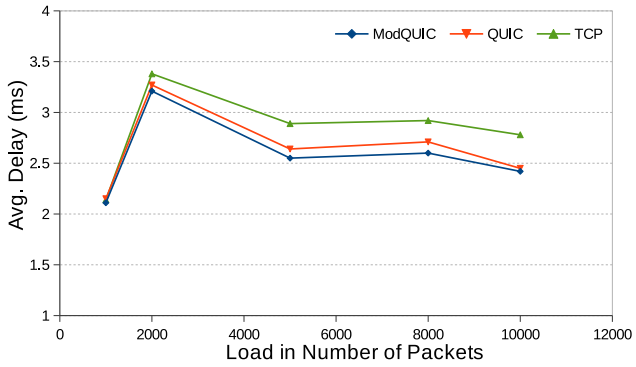
window update instead of based on ACK reception time; which provides sufficient ground to increase window size to specified level. This results in throughput improvement observed in Table 2/Figure 7.

In ModQUIC, window update has been done with every ACK, whereas in QUIC, window update is based on ACK reception time analysis. In ACK reception time analysis, if new ACK reception time is greater than previous ACK reception time then there is no window update. This is the threat in updating congestion window size in QUIC. Further in QUIC, extra time is required to do the ACK received time analysis and to send window update frame separately. We removed this threat and window update delay by attaching window update frame to ACK frame. This improves the overall performance of ModQUIC compared to QUIC and TCP resulting in optimum of the bandwidth.

Table 3 Performance comparison w.r.t. delay

No. of packets	*Throughput (Mbps)*			*% improvement of ModQUIC over QUIC*
	ModQUIC	*QUIC*	*TCP*	
1,000	2.11	2.15	2.14	1.86
2,000	3.21	3.27	3.38	1.83
5,000	2.55	2.64	2.89	3.40
8,000	2.60	2.71	2.92	4.06
10,000	2.42	2.45	2.78	1.22

Figure 8 Performance comparison w.r.t. delay (see online version for colours)



Updating window size dynamically for each ACK ensures the delivery of data within optimum time. This reduces number of retransmissions as minimal loss of data is observed. The data loss check reduce on redundancy in packet delivery. Which results into more number of packets delivered within time compared to existing system. By this way average end to end packet transmission delay is reduced shown in Table 3/Figure 8.

In ModQUIC/QUIC, loss-based congestion control mechanism has been employed. The response to loss is a cubic function, which is slow at the beginning and growth is exponential later. This shows performance of ModQUIC, QUIC has been improved with respect to loss. However as greater bandwidth has been occupied by ModQUIC/QUIC compared to TCP, which is unfair. Higher the percentage of ACKs generated, more are the number of packets received. Validation of said analysis has been presented in Table 4/Figure 9. When the loss is 0%, TCP outperforms due to it's initial aggressiveness whereas ModQUIC and QUIC performance is almost similar. TCP performance gradually decreases with respect to loss rate, whereas in ModQUIC and QUIC, due to multiplexed streams and out of order delivery, even in lossy links the performance is superior. Performance of ModQUIC is better than QUIC due to bandwidth occupancy limitation which in turn depends upon the window size used. Even though slow start is avoided in QUIC, its default window size is updated only with reference to an analysis of previously sent packet's success and their rate of transmission. This improves reception of ACKs, and the graph shows that ModQUIC outperforms over QUIC and TCP. In ModQUIC, maximum bandwidth utilisation has been observed due to successful reception of ACK which responsible for window update.

Table 4 Performance comparison w.r.t. loss

Loss rate (%)	*ACKs generated (%)*		
	ModQUIC	*QUIC*	*TCP*
0	91.40	89.30	98.23
1	90.32	88.67	93.57
2	85.12	82.34	85.36
5	97.76	93.82	85.47
8	99.56	98.26	81.88
10	97.98	96.71	80.19

Figure 9 Performance comparison w.r.t. loss (see online version for colours)

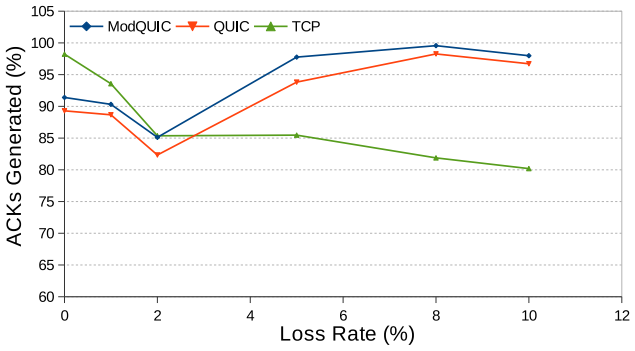


Table 5/Figure 10 shows that for lossless bottleneck link of 2 Mbps all three flows are closely competing with each other. As loss rate increases link becomes congestive and as a reaction TCP reduces data rate gradually but for ModQUIC and QUIC base protocol is UDP, performance is better. Whereas for ideal condition [sufficient bandwidth (10 Mbps) and lossless link] TCP is dominating as packet pacing becomes overhead for ModQUIC and QUIC. Once loss rate increases TCP performance suddenly drops down below ModQUIC and QUIC observed in Table 6/Figure 11.

Table 5 Data rate achieved for 2 Mbps link w.r.t loss

Loss rate (%)	*Avg. data rate achieved (Mbps)*		
	ModQUIC	*QUIC*	*TCP*
0	1.891	1.842	1.920
1	1.853	1.815	1.913
2	1.836	1.794	1.851
3	1.815	1.776	1.785
4	1.772	1.732	1.708
5	1.725	1.681	1.643

Figure 10 Data rate variation w.r.t. loss for 2 Mbps link (see online version for colours)

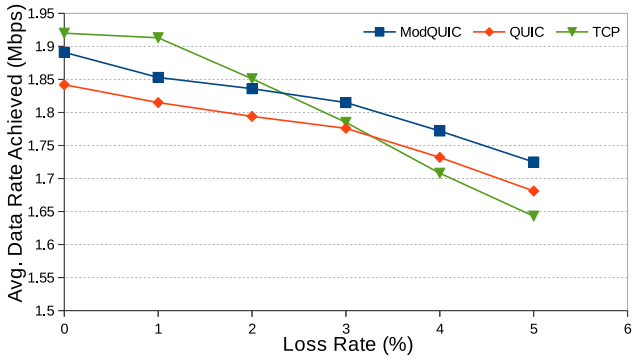
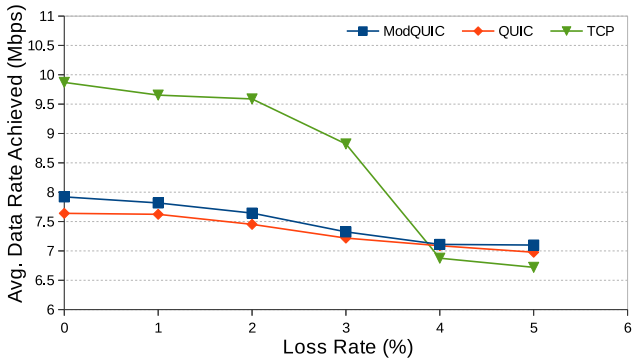


Table 6 Data rate achieved for 10 Mbps link w.r.t loss

Loss rate (%)	*Avg. data rate achieved (Mbps)*		
	ModQUIC	*QUIC*	*TCP*
0	7.920	7.640	9.870
1	7.817	7.623	9.654
2	7.644	7.452	9.588
3	7.324	7.218	8.822
4	7.111	7.088	6.876
5	7.100	6.977	6.720

Figure 11 Data rate variation w.r.t. loss for 10 Mbps link (see online version for colours)



4.4 Validation of results

To validate and check consistency of the obtained results, linear regression model, R^2 (R-squared) has been employed, which is a statistical analysis indicating percentage of variance given in equation (5). Regression analysis is a set of statistical processes for estimating the relationships among variables. More specifically, regression analysis helps us to understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed.

$$R^2 = \frac{Variance\ explained\ by\ the\ model}{Total\ Variance} \times 100 \qquad (5)$$

where, R^2 value ranges between 0% to 100%.

A 0% represents suggested model is not consistent with respect to performance parameters, whereas 100% indicates model is perfect with respect to performance parameters. Larger is the R^2 value, better is the regression model that fits to observations. Table 7 shows R^2 values for performance comparison of ModQUIC with QUIC and TCP. Overall observations are seen to fit to the regression model. However the values of 70% and 33% are lesser with respect to the desired output. It may be noted that, the lower R^2 value indicate ModQUIC performance to be improving significantly compared to TCP. In delay and loss result analysis, ModQUIC and QUIC are almost following similar pattern whereas ModQUIC and TCP are seen to drift from each other. In ModQUIC there is a higher reduction in delay compared to TCP with respect to number of packets in flight. TCP performance seen to be very poor with respect to loss compare to ModQUIC.

Table 7 R^2 values for ModQUIC performance validation

Parameters	*R^2 value (%)*	
	ModQUIC V/s QUIC	*ModQUIC V/s TCP*
Throughput	97	92
Delay	96	70
Loss rate	96	33
Data rate for 2 Mbps link w.r.t. loss	99	96
Data rate for 10 Mbps link w.r.t. loss	98	89

5 Conclusions

To improve overall network throughput, Google developed QUIC protocol to compete TCP's dominance. In order to enhance the throughput and reduce transmit delay (latency) further ModQUIC has been proposed. In this work, Chromium server-client model testbed environment has been used and performance tested. The presented results in the form of comparative analysis of ModQUIC, QUIC and TCP using parameters throughput, delay and behaviour with respect to loss rate, for limited and sufficient bandwidth condition. The observed improvement for ModQUIC in throughput over QUIC and TCP are 35.66% and 51.93% respectively. Whereas marginal reduction in delay compared to QUIC and significant over TCP has been observed. It has been also

observed that for lossy link TCP shows poor performance. However the performance of ModQUIC and QUIC found to be stable. For high-speed link (sufficient bandwidth) QUIC and ModQUIC act as a performance bottleneck. Results were validated with the help of R^2 regression model. As an extension of this work, performance testing of ModQUIC for multiple clients with fairness property can be done.

References

Afanasyev, A., Tilley, N., Reiher, P. and Kleinrock, L. (2010) 'Host-to-host congestion control for TCP', *IEEE Communications Surveys and Tutorials*, Vol. 12, No. 3, pp.304–342.

Aggarwal, A., Savage, S. and Anderson, T. (2000) 'Understanding the performance of TCP pacing', in *Proceedings IEEE Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2000*, IEEE, Vol. 3, pp.1157–1165.

Biswal, P. and Gnawali, O. (2016) 'Does quic make the web faster?', in *2016 IEEE Global Communications Conference (GLOBECOM)*, IEEE, pp.1–6.

Carlucci, G., De Cicco, L. and Mascolo, S. (2015) 'Http over udp: an experimental investigation of quic', in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ACM, pp.609–614.

Cook, S., Mathieu, B., Truong, P. and Hamchaoui, I. (2017) 'Quic: better for what and for whom?', in *2017 IEEE International Conference on Communications (ICC)*, IEEE, pp.1–6.

Das, S.R. (2014) *Evaluation of QUIC on Web Page Performance*, PhD thesis, Massachusetts Institute of Technology.

Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Karim, T., Katz-Bassett, E. and Govindan, R. (2016) 'An internet-wide analysis of traffic policing', in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, ACM, pp.468–482.

Grigorik, I. (2017) *High Performance Browser Networking: What every Web Developer should know about Networking and Web Performance*, O'Reilly Media, Inc., Sebastopol, California.

Gizis, A. (2016) *Taking Google's Quic for a Test Drive* [online] http://www.connectify.me/blog/taking-google-quic-for-a-test-drive/.

Ha, S., Rhee, I. and Xu, L. (2008) 'Cubic: a new TCP-friendly high-speed TCP variant', *ACM SIGOPS Operating Systems Review*, Vol. 42, No. 5, pp.64–74.

Hamilton, R., Iyengar, J., Swett, I. and Wilk, A. (2016) 'Quic: a udp-based secure and reliable transport for http/2', *IETF, Draft-tsvwg-quic-protocol-02*.

Hassan, M. and Jain, R. (2003) *High Performance TCP/IP Networking*, Vol. 29, Prentice Hall, , Upper Saddle River, New Jersey, USA.

Iyengar, J. (2016) *Quic: Redefining iInternet Transport*, Google Inc [online] https://docs.google.com/presentation/d/15e1bLKYeN56GL1oTJSF9OZiUsI-rcxisLo9dEyDkWQs.

Iyengar, J. and Thomson, M. (2017) 'Quic: a udp-based multiplexed and secure transport', *Draft-ietf-quic-transport-01 (work in progress)*.

Kakhki, A.M., Jero, S., Choffnes, D., Nita-Rotaru, C. and Mislove, A. (2017) 'Taking a long look at QUIC', in *Proceedings of the 2017 Internet Measurement Conference*.

Langley, A. and Chang, W-T. (2013) *Quic Crypto* [online] http://tinyurl.com/lrrjyjs.

Leith, D. and Shorten, R. (2004) 'H-tcp: TCP for high-speed and long-distance networks', in *Proceedings of PFLDnet*.

Luckie, M., Dhamdhere, A., Clark, D., Huffaker, B. et al. (2014) 'Challenges in inferring internet interdomain congestion', in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ACM, pp.15–22.

Megyesi, P., Krämer, Z. and Molnár, S. (2016) 'How quick is quic?', in *2016 IEEE International Conference on Communications (ICC)*, IEEE, pp.1–6.

Ross, S.M. (1996) *Stochastic Processes*, 2nd ed., Wiley Series in Probability and Mathematical Statistics, Wiley, India.

Roskind, J. (2012) *QUIC: Design Document and Specification Rationale*, Internet-Draft, Intended status: Informational, Internet Engineering Task Force.

Swett, I. (2015) *Quic Loss Recovery and Congestion Control*, Internet-Draft, Intended status: Informational, Internet Engineering Task Force.

Swett, I. and Iyengar, J. (2015) 'Quic loss recovery and congestion control', *Internet-Draft, Intended status: Informational, Internet Engineering Task Force*.

Srivastava, A. (2017) *Performance Analysis of QUIC Protocol under Network Congestion*, PhD thesis, Master thesis, Worcester Polytechnic Institute.

Xu, L., Harfoush, K. and Rhee, I. (2004) 'Binary increase congestion control (bic) for fast long-distance networks', in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, IEEE, Vol. 4, pp.2514–2524.