Performance analysis of Google's Quick UDP Internet Connection Protocol under Software Simulator

Saif Talib Albasrawi
Education Collage /Misan University, Iraq
saiftalib@uomisan.edu.iq

Abstract: The QUIC protocol (Quick UDP Internet Connection) is a transport based over UDP Protocol (User Datagram Protocol) that provides safe, reliable and fast service on the Internet. Google proposed it to solve the problem of network delay. It is efficient, fast, and takes up fewer resources. The QUIC gathers the advantages of both TCP and UDP. QUIC is a user-level protocol running on top of UDP, which considered by IETF (Internet Engineering Task Force). Google assumes the response time of page load was short so that end-user performance was better. In this paper, the results presented on a local test using NS-3 (network simulator version 3) that allows QUIC output to test and design choices and possible limitations present the description of the functionalities and the main assumptions of the internet QUIC.

KEYWORDS
TCP protocol, UDP protocol, QUIC protocol, NS-3.

1 INTRODUCTION
Internet applications require reliable, fast communication and security. The TCP (Transmission Control Protocol) also provides the reliability and protection achieved by the TLS (Transport Layer Security) protocol. These protocols work in clients (user node) and servers (end nodes). Although the network infrastructure retains the high speed, it has to conserve for the applications. TCP and TLS have a critical role in user-comprehending performance [1].

QUIC is a network transport protocol; Google suggested this as a user-level protocol running over UDP instead of TCP, thus removing the necessity for the TCP protocol's initial handshake function. It runs its encryption scheme, which is comparable to TLS, combines link establishment and key agreement into 1 RTT only. In any case, QUIC can start a connection in 0 RTT, straight away sending encrypted application information or data to the server, when it already has in its cache the server declaration (from the previous connection). In any case, if QUIC already has the server declaration in its cache (from the previous connection), it can start a link in 0 RTT, send immediately encrypted application information or data to the server. Few research papers typically focused on the QUIC as transport for HTTP [2]. QUIC only used by its services on Google's servers (G mail, Search, YouTube, etc.), and only Chrome and Chromium browsers initially support the QUIC protocol [3]. Lastly, NS-3 implementations and existing execution of congestion control algorithms rely on the configuration of the NS-3 TCP socket, which cannot reuse without the native implementation. In this manner, in the spirit of expressing the responsibility for implementing and updating the protocol, QUIC
implementations for NS-3 would support the research community in networking due to their usability. Also, the QUIC stack intended to reuse the various TCP congestion management implementations within QUIC. To compare different congestion management designs for QUIC directly conceivable, while actual implementations give the New Reno adjustments only [4]. In the remainder of this paper, an outline of the significant highlights of QUIC provided in Sec. 2. Then, a description of the NS-3 implementation in Section 3, with more information on the code structure, QUIC compatibility with TCP and missing QUIC Internet Drafts components, in Sec. 4, presents examples of QUIC performance evaluation. At last, the results concluded in Sec. 5.

2 Criteria of QUIC Transport Protocol
QUIC is a protocol for transport layers that aims to increase performance. It based on UDP that allows fast deployment of QUIC changes. Figure 1 shows how an application differs between TCP and QUIC [2].

Two major design decisions make it possible to improve efficiency compared to TCP, QUIC integrates cryptography and handshake transportation to minimize setup latency and by default to provide a secure channel. To do this, it provides three types of connection setup [5, 6].

- **Initial handshake (Initial handshake 1-RTT):** Firstly, the client have no any server details. The client begins with a message about Client Hello (CHLO), which will discard with a REJ message by the server. It requires a server configuration with a long-term server authentication chain and a time stamp. Presently, the customer can send a complete message to CHLO containing the initial labels (tags) and the REJ messages. If the handshake is efficient, the server will responding to an encrypted Hello message from the server (SHLO). The hello message for the server contains the ephemeral public value for the server that used to measure the ephemeral session key.

- **Repeat handshake (0-RTT):** In any past connection establishment, the client has only observed the REJ post. Which stored the labels (tags) from a REJ message in order to complete the Client Hello message (CHLO). Again, if the handshake is successful, the server will respond to an encrypted SHLO response. Through using the initial shared key, the two meetings will calculate the ephemeral keys to send and receive further messages. From that point, when the client wishes to reach 0-RTT latency, the request must encrypt with primary keys and send before receiving a server response. The server also stores the client's nonce and its public interest, which determines the shared key, to accomplish this.

- **Rejected 0-RTT (Failed 0-RTT):** The server reacts with the REJ message when the information on the server has expired in complete CHLO. In this case, the 0-RTT try failed and the handshake proceeds as though it were an initial handshake.

The second performance-enhancing structure decision addresses the head of a line blocking issue. It occurs when a packet is lost during transit and has to retransmit. TCP's efficient service

![Figure 1. QUIC in the traditional stack](image-url)
guarantees to have the packets delivered in the same order when they have sent. It makes TCP traffic vulnerable to header blocking; on delivery of the missing packet, subsequent packets must withstand. For every problem, QUIC uses different sources of lightweight structured abstractions. Each stream cut into frames. There are multiple forms of frames. Start with the standard stream frames contain data used to create connections (e.g., CHLO, REJ). It uses fixed stream 1. First, acknowledgment frames, which will address later, inform the sender of the successful delivery of packets. There is a congestion management unit, which is not included in our environment since the local network has only one single user. Ultimately, two types of frames are required to close this connection; they will be addressed later in section 4 [5].

![Image: Outline of Various QUIC handshakes [6]](image)

**Figure 2. Outline of Various QUIC handshakes [6]**

When a packet gets lost, it only affects certain streams that hold data in this packet. Subsequent data obtained for reassembly and delivery to the client.

2.1 *Important features of QUIC protocol include:*  
- **Communication Implementation Latency:** QUIC protocol integrates cryptography and handshake transport to reduce the amount of round trips needed for communication. It offers a cached client code that has reused to communicate with a previously considered server. It reduces the necessity for a new handshake.  
- **Multiplexing:** QUIC multicast consists of streams, which transport data independently. Data for each stream in the specified frame sent using the stream identifier ID. A QUIC packet can consist of one or more frames.  
- **Forward Error Correction:** QUIC supports FEC, where an FEC packet will contain the parity of the packet forming the FEC group. This function enabled or disabled if necessary. It allows it to recover the contents of a lost packet in an FEC group.  
- **Connection Migration:** QUIC connection defined by the 64-bit connection identifier instead of 4 tuples from the IP addresses of source and destination and the essential communication port numbers. Thus, for example, if a device changes the Internet connection, if the IP addresses or NAT (Network Address Translation) connections change, the QUIC connection reused. QUIC allows encryption authentication of a relaying client, so using the session key, the client will encrypt and decrypt packets. [7, 8].
3. QUIC simulation
This section explains a simulation of the software. The benefit of QUIC protocol, and how the main classes of Quic protocol (internet module) can modify in the NS-3 to match UDP and TCP. Duplicating the isolation of congestion control from the primary socket and the presence of stand-alone buffer classes [9], while adding new components to TCP with representation for the novelties. As in the TCP execution, each client's Quic sockets models will instance a single Quic SocketBase entity, while the server will branch a new socket for all incoming connections. A Quic SocketBase object receives and transmits acknowledgments and packets, accounts for retransmissions, performs connection-level flow and congestion control, concerns the initial handshake, exchanges transport parameters, and manages the state machine and the life cycle of a QUIC link. The class Quic SocketBase provides pointers to many other related elements including:

A) Quic SocketTxBuffer and Quic SocketRxBuffer introduced socket transfer and reception buffers, respectively.

B) Quic SocketState object expanding TCP Socket State [9] with more variables used by the state and congestion control machines.

An Object extends the TCP CongestionOps class, which incorporates congestion control and makes it compliant with TCP's congestion control implementation. The QUIC socket connected to the main UDP socket via the Quic L4Protocol object, which also manages the initial development of a Quic SocketBase object, triggers the joining UDP socket, and the distribution of packets between the Quic SocketBase and UDP sockets. Instead a stream shown by the Quic StreamBase class which extends the basic Quic Stream class. It buffers application information, implements flow control at stream-level, and transmits the data received to the application. In addition, the Quic StreamBase object has pointers to transmit and receive the buffers to the full socket, which performed in the classes Quic StreamTxBuffer and Quic StreamRxBuff.

Various Quic StreamBase Objects connected via an entity to a single Quic SocketBase belong to the Quic L5Protocol class. The Quic SocketBase includes a pointer to an object Quic L5Protocol, and the last one carries a reference vector to different instances of Quic StreamBase. The Quic L5Protocol class establishes and configures the flows and issues of transmitting packets through flows and sockets to transmit and receive.

3.1 Packets, Frames and Headers
A QUIC packet consists of a header and a payload, as shown by projects on the QUIC Internet. The QUIC packet encapsulated in a wire-transmitted UDP datagram. A packet header provided by the class Quic Header that extends the class ns-3 Header. When connected to the ns-3 packet, it deals with serialization and deserialization of the header and represents data transmitted by a QUIC header. One of the key novelties concerning the QUIC header structure's conventional TCP headers is that it can have two separate formats (long or short), as shown by the data calculation that should be shared. The passage in the packet number header can also have a variable length for short headers (one, two, or four bytes), depending on the number of bytes required to represent the packet number. For long headers (17 bytes), used to exchange messages that are essential to the link's lifecycle (such as initial handshake and version negotiation). Short headers (2-13 bytes) used in data packets and acknowledgments. The Quic Header class offers different static methods that can use to make various header types (long, short, for handshake message) and to set significant parameters. Alternatively, the Quic packet payload consists of several frames, every frame mapped to a stream and / or a control process for each frame also conveys an extra subheader enforced by the Quic Sub-header class, and indicates the frame type. Data frames properly connected to flow via a flow identifier, and their subheaders can convey frame size. Instead, the control frames used to perform specific actions; the ACK frames are the most control frames in the NS-3 Quic code; implement them as shown in [10,11]. And it used to recognize packets that received at the endpoint of
the link. Each ACK assigns the most significant number of accepted sequences and, when finding gaps in the received packet list, can hold multiple selective accepting blocks. In the ACK, the interval with the most important sequence number between the accepted packet receptions can list with the time it sent. The QUIC-protocol packet structure described in Figure 3 [12].

![Diagram](image)

**Figure 3.** Packet structure of QUIC protocol

3.2 **QUIC Buffers**

- **In socket level,** the Quic SocketTxBuffer transmission buffer stores flow or control frames which Quic SocketBase passed into the socket and returned packets of the desired size. It comprises two separate types of packets, the first type to which it is still not transmitted, and the second type to which it is transmitted (until it is recognized) if retransmission is required. Each element stored in a Quic SocketTxItem list relates the intelligent pointer to the timing of the ACK receipt, the packet with sequence number and some flags that mark the packet when lost [13]. Additionally, it can separate and assemble frames depending on the specific packet size the socket requires. In the first transmission, when a packet is a transfer to the socket, the relevant frames removed from the unsent list; the entire packet has entered the sent list. In this case, it can control ACK receipt correctly, release packets exactly received, measure the number of bytes and handle retransmissions [14].

- **In stream level,** Quic StreamTxBuffer manages the sender side buffering. The transmission buffers store application packets until they accept it, but it does not retransmit, which based on the QUIC project's internet-draft, which stored all QUIC packets at the socket stage. In the stream, the packets buffer from the start of the stream marked by an offset. Since the received byte is already out of order, it will deposit to the stream buffer before the packet containing the missing byte received. As the stream buffer gets multiple bytes in an incorrect order, it passes them to the Quic SocketRxBuffer; it stores the received data packets released before the client requests them.

The stream includes the Finish bit (meaning the transmission has ended), which will recover the entire amount of bytes obtained in the stream level buffer. The concept that buffers in either Quic StreamBase or Quic SocketBase are always initialized [12, 15]. The buffers in this paper follow the TCP implementation method in ns-3, set the buffer size, the number of bytes that can manage buffering before rejecting a packet by using the socket features of those classes, and the stream buffers.
3.3 Retransmission Process
The QUIC retransmission mechanism works at the socket-level. It retransmits complete packets consisting of one or many frames that have lost by increasing the reception flow and ACK transmission used in QUIC implementation. In this case, ACK reception managed by the Quic SocketBase class during the RecAckFrame process and Quic L5Protocol begins when an ACK frame is in the received packet. The ACK frame periodically defines the most significant recognized number of packets, probably includes a few additional ACK blocks, which indicate missed packets [16]. The socket must pass the information to the Quic SocketTx buffer associated with it, which will mark it as missing or recognized, and restore the list to the socket. The socket performs proper congestion control activities at this stage by upgrading the socket status in case of loss or through the congestion window and activating the retransmission process, retransmissions in the QUIC protocol connected to the new one, monotonically-sequence numbers. Therefore, Quic SocketBase increases sequence number value by one each retransmission and advises the socket buffer. The last transfers the sent packets identified as missing back to the buffer that has not yet transmitted, updating their packet number. Finally, Quic SocketBase retransmits and forwards the packets to Quic L4Protocol [17].

3.4 QUIC compatibility with TCP congestion control
The modularity algorithm for congestion control according to the primary socket code mentioned in [9], This is one of the essential features of NS-3. It accomplished by using a specific object, which expands the congestion control functionality of a class with the fundamental TCP CongestionOps. Congestion management algorithms have introduced and joined to the Internet node. Vegas, for example, of TCP's NewReno. Since the legacy mode can use in Quic SocketBase. To push the congestion window for the QUIC link it can use TCP algorithms to manage congestion. It can use specific algorithms that have used additional details. Supported only for QUIC by the socket QUIC and the development of congestion management techniques. [18]. Compatibility with legacy TCP congestion control algorithms achieved by keeping the instance of TCP CongestionOps as the primary object of congestion control in Quic SocketBase. Later introduced the new Quic CongestionControl class, which extends TCP NewReno. Quic CongestionControl has other methods which include additional information that QUIC Internet-Draft defines in the congestion control algorithm (for example, packet transmission, more accurate RTT information) if required. [19].

3.5 Connection Structure
The connection of QUIC is one communication between two endpoints, each defined by a combination of IP addresses and UDP ports in a specific way [11]. The QUIC system introduced an interactive internet-draft with open connections and introduced client-server version matching and handshaking. In addition, the procedure implemented models handshake but did not explicitly encrypt. The QUIC will handshake based on the combined endpoint date in two separate ways. Upon recently authentication of the endpoints, 0-RTT handshaking can use. It models by attaching a list of checked endpoints to Quic L4Protocol that tested before the first packet sent and at another endpoint when it received. In addition, 0-RTT was added in Quic L4Protocol using the Handshake attribute 0-RTT. If the 0-RTT not enabled, then the socket reverts with the original client message to a 1-RTT handshake. Lastly, if the two endpoints in a version differ, a 2-RTT handshake will use to stabilize the point-to-point release of QUIC [20, 21].

3.6 Applications and Testing
Finally, an appropriate set of unit tests performed; it will expand in future releases. The test suite (quic-tx-buffer) first tests the correctness of the buffering behavior of stream and socket transmission, implemented by Quic StreamTxBuffer and Quic SocketTxBuffer. The test checks various inserts for both stream and socket and removes events from buffers, including tests where the insert failed due to the presence of the entire output buffer and also, the packet must reinsert in the stream buffer. In addition, socket checking in case of retransmission verifies the behavior. Likewise, the second test
community (Quic Rx Buffer) performs on the socket buffers to add and remove operations and stream reception. The third set of tests (the quic header) finally checks Quic Header and Quic Subheader implementations. In addition, to directly use the streaming multiplexing features, which introduced the adaptation of Quic protocol’s UDP client-server (Client and Server) applications, allowing the API to send to the socket using the stream identifier. So, the data scheduled through the available streams.

4 Test Results
Now, it presents a set of preliminary results for testing NS-3 QUIC implementation using various congestion control algorithms in both the legacy and non-legacy modes. By using standard dumb-bell topology, also used to test TCP performance [22] when a common 5 Mbps bottleneck is used between clients and servers. In our example, the minimum RTT is 50 milliseconds used to implement this experiment has been a quic-variants-comparison, which has changed from TCP. For two old congestion management algorithms (i.e., New Reno and Vegas), with the non-legacy alternative New Reno adapts the congestion avoidance stage to a slightly different congestion window. The development of congestion windows corresponds to the predicted behavior of the BDP product (Bandwidth-Delay Product) algorithms with two steady-state flows (20 kB). Furthermore, when comparing the actions associated with delay, congestion (i.e., Vegas) and loss (i.e., New Reno and QUIC), you can see that Vegas has managed to save less RTT, as planned. Lastly, congestion management of New Reno and QUIC showed similar patterns. The latter distinguished at the congestion avoidance level by improved window width.

4.1 Comparison of performance between TCP and QUIC
This test performed to compare the effect of various network connection parameters on TCP and QUIC transmission. The purpose of this test is to examine the behavior of QUIC, which is the same or better than TCP, by comparing protocol interactions with different network bandwidth. Then to run the same test, but with TCP, the bytes sent had the same file size as used in the QUIC test. Traffic shaping also performed using the same test in the QUIC protocol. The throughput test in Figure 4 and Figure 5 shows the different throughput of TCP and QUIC with various Bandwidth and RTT.

![Figure 4. Comparison between TCP and QUIC](image-url)
Figure 5. Test 0-400ms Round Trip Time (RTT), 50Mbps and 0% loss

5 Conclusions
In this paper, the original QUIC implementation introduced on the base of NS-3, which is the new IETF standardized transport protocol and represents the majority of Internet traffic. The QUIC application mentioned in this paper expands the structure of the TCP code in NS-3 with functions that differentiate QUIC, such as stream multiplexing, and initial handshakes with low latency. In addition, improve the QUIC socket framework so that both the latest QUIC-only congestion control algorithms and the legacy TCP can plugin. Testing the implementation using simulations, a dumb-bell topology showing the standard behavior of congestion control algorithms also compared their performance with a non-legacy internet.
References

[1] Rasool A, Alpar G, and de Ruiter J 2019 State machine inference of QUIC arXiv preprint arXiv:1903.04384.

[2] Carlucci G, De Cicco L and Mascolo S 2015 HTTP over UDP: An experimental investigation of QUIC in Proc. of the 30th Annual ACM Symp. on Applied Computing-SAC Vol. 15 pp 609–614

[3] Cook S, Mathieu B, Truong P and Hamchaoui I 2017 QUIC: Better for what and for whom? IEEE Int. Conf. on Communications (ICC) pp 1-6

[4] Kakhki A, Jero S, Choffnes D, Nita-Rotaru C and Mislove A, 2017 Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols in Proc. of Internet Measurement Conf. pp 290-303

[5] Lychev R, Jero S, Boldyreva A and Nita-Rotaru C 2015 How secure and quick is QUIC? Provable Security and performance analyses IEEE Symp. on Security and Privacy (San Jose: IEEE) pp 214-231

[6] Langley A, Riddoch A, Wilk A, Vicente A, Krasic C, Zhang D, Yang F 2017 The quic transport protocol: Design and internet-scale deployment in Proc. of the Conf. of the ACM Special Interest Group on Data Communication pp 183-196

[7] Das S 2014 Evaluation of QUIC on web page performance Doctoral dissertation, Massachusetts Institute of Technology

[8] Stone C, Chothia T and de Ruiter J 2018 Extending automated protocol state learning for the handshake. In European Symp. on Research in Computer Security (Springer Cham) pp 325-345

[9] Casoni M and Patriciello N 2016 Next-generation TCP for ns-3 simulator Simulation Modelling Practice and Theory vol 66 pp 81-93

[10] Iyengar J and Swett I 2018 Quic loss detection and congestion control Internet Engineering Task Force.[Online]. Available from: https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-07 [Accessed 19 Mar 2020]

[11] Iyengar J and Thomson M 2018 Quic: A udp-based multiplexed and secure transport Internet Engineering Task Force. [Online]. Available from: https://tools.ietf.org/html/draft-ietf-quic-transport-17 [Accessed 19 Mar 2020]

[12] Bishop S, Fairbairn M, Norrish M, Sewell P, Smith M and Wansbrough K 2005 Rigorous specification and conformance testing techniques for network protocols, as applied to TCP, UDP, and sockets in Proc. of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications vol. 35 pp 265–276

[13] Ekiz, N, Rahman A and Amer P, 2011 Misbehaviors in TCP SACK generation ACM SIGCOMM Computer Communication Review, vol. 41(2) pp16-23

[14] Thomson M and Turner S 2019 Using TLS to Secure QUIC [Online]. Available from: https://quicwg.org/base-drafts/draft-ietf-quic-tls.html [Accessed 19 Mar 2020]

[15] Shade R and Warres M 2016 HTTP/2 Semantics Using The QUIC Transport Protocol IETF. [Online]. Available: https://tools.ietf.org/html/draft-shade-quic-http2-mapping-00 [Accessed 19 Mar 2020]

[16] Megyesi P, Kramer Z and Molnar S 2016 How quick is QUIC? in IEEE Int. Conf. on Communications pp. 1-6

[17] Carlucci G, De Cicco L and Mascolo S 2015 HTTP over UDP: an Experimental Investigation of QUIC in Proc. of the 30th Annual ACM Symp. on Applied Computing pp 609-614

[18] Nguyen T, Gangadhar S, Rahman M and Sterbenz 2016 An Implementation of Scalable, Vegas, Veno, and YeAH Congestion Control Algorithms in ns-3 in Proc. of the Workshop on ns-3 pp 17-24
[19] Hamilton R, Iyengar J, Swett I and Wilk A 2016 Quic: A UDP-based secure and reliable transport for http/2.IETF. [Online]. Available from: https://tools.ietf.org/id/draft-tsvwg-quic-protocol-02.html [Accessed 19 Mar 2020]

[20] Fischlin M and Gunther F 2014 Multi-stage key exchange and the case of Google's QUIC protocol in Proc. of the ACM SIGSAC Conf. on Computer and Communications Security pp 1193-1204

[21] Ruth J, Poese I, Dietze C and Hohlfeld O 2018 A First Look at QUIC in the Wild in Int. Conf. on Passive and Active Network Measurement (Springer Cham) pp 255-268

[22] Fiterau P, Janssen R and Vaandrager F 2016 Combining model learning and model checking to analyze TCP implementations in International Conference on Computer-Aided Verification (Berlin: Springer Cham) pp 454-471