The QUIC Transport protocol: Motivation, Design, and Benefits

Vi Luan Dang - 103802759

Faculty of Science, Engineering and Technology

Swinburne University of Technology

Ho Chi Minh, A35 Bach Dang, Tan Binh District

Abstract

QUIC (Quick UDP Internet Connections) is a new transport protocol built on UDP that aims to address the limitations of TCP in HTTP/2 transmission (Yong, Li, & Cong, 2012). The design of QUIC is motivated by the need for reduced latency and faster connections for web applications, as well as the shortcomings of TCP over wireless network. This paper explores the design and implementation of QUIC. We discuss its novel features including connection establishment, congestion control, stream scheduling, and security. Several disadvantages of applying QUIC will also be discussed.

Keywords: QUIC, transport protocol, UDP, latency, congestion control, stream multiplexing, security.

1. Introduction

The web has become a platform for interactive, latency-sensitive applications such as online gaming, web conferencing, and remote collaboration tools. However, the Transmission Control Protocol (TCP), the dominant transport protocol that underlies web traffic, was designed for more traditional client-server applications that trade latency-throughput for reliability. Therefore, TCP require one Round-trip Time (RTT) of latency due to its handshake. To solve these matters, a new transport protocol called QUIC has been introduced, it is built on top of UDP, and its design is integrated with the best practices of multiple existing protocols including TCP, TLS, and HTTP/2 (Yong, Li, & Cong, 2012).

QUIC stands for Quick UDP Internet Connections and was developed by Google (Puneet, 2020). The architecture of QUIC is as follows:

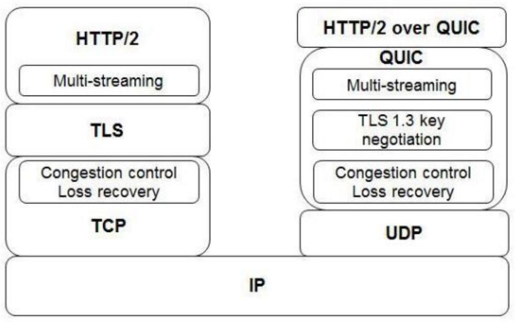


Figure 1. QUIC Architecture

As shown in the figure above, QUIC sits on top of UDP to be compatible with all the middleboxes (Puneet, 2020), and therefore eliminates the limitations of slow-connection establishment due to three-way handshake. Additionally, QUIC also compensates for UDP shortcomings of lacking encryption, multiplexing, and congestion control.

2. Motivation

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information system (Fielding et al., 1996). It established the basic patterns of client-server interaction and request-response messaging over the Internet that have formed the foundation for the modern World Wide Web. The following figure will demonstrate how HTTP/1.0 works:

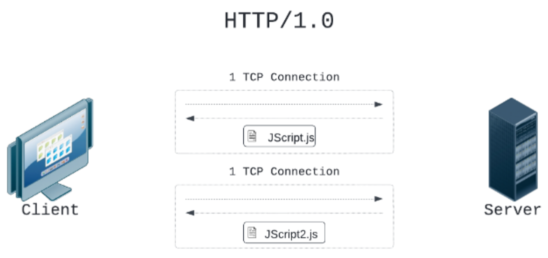


Figure 2: HTTP/1.0 connection.

According to the figure above, HTTP protocol is based on a request/response paradigm(Fielding et al., 1996). A client establishes a TCP connection to a server and sends a request message. The server processes the request and returns a response message. The client then interprets and displays the response content. After the response is completed, the TCP connection is closed. While HTTP 1.0 introduced the core patterns of client-server web interactions that still persist today, it only supported basic request methods and did not use persistent connections, limiting its performance for some use cases. The lack of persistent connections, compression ,and encryptions are some of the key limitations of HTTP/1.0.

HTTP/1.1 was introduced in 1997 (Fielding et al., 1997) as an update to HTTP/1.0. It improved the original protocol in many ways, most noticeable is persistent connections and pipelining. With such improvement, HTTP/1.1 has successfully reduced latency and delivered essential upgrades that made the web faster and more robust (Fielding et al., 1997). The following figure will illustrate how HTTP/1.1 works:

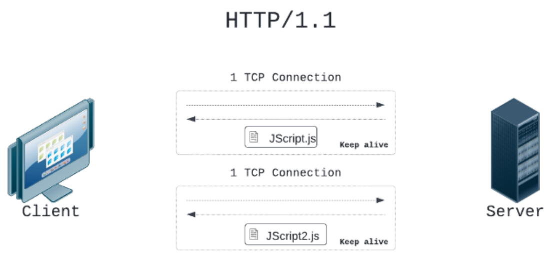


Figure 3: HTTP/1.1 Connection.

Persistent connections in HTTP/1.1 is introduced via "keep alive" mechanism, allowing multiple requests and responses to be sent over the same TCP connections. That is whenever a TCP connection between the client and server has successfully delivered its content, the TCP connection will remain open and thus allow another package to be delivered serially (Fielding et al., 1997). This eliminates the overhead of establishing a new connection for each new request. HTTP/1.1 also added pipelining, enabling multiple requests to be sent without waiting for each response. This further reduces latency.

While HTTP/1.1 delivered significant improvements over HTTP/1.0, including persistent connections, pipelining and new methods, it still has some disadvantages (Rahman & Rahman, 2017). Although persistent connections and pipelining help reduce latency by eliminating the overhead of creating new connections, they can also lead to congestion and head-of-line blocking problems. While being able to send multiple requests is convenient, HTTP/1.1 can only send them serially, therefore the server and the client can only execute a single request responses exchange at any given moment. Moreover, if a slow request is pipelined before faster requests, the faster requests have to wait for the slow one to finish, delaying the responses (Rahman & Rahman, 2017).

Over the years, the need for a better, more advanced web protocol to replace HTTP/1.1 has driven the development of HTTP/2. HTTP/2 was developed to address the limitation of HTTP/1.1 (Wang et al., 2019). the following figure will give a thorough concept of how HTTP/2 works:

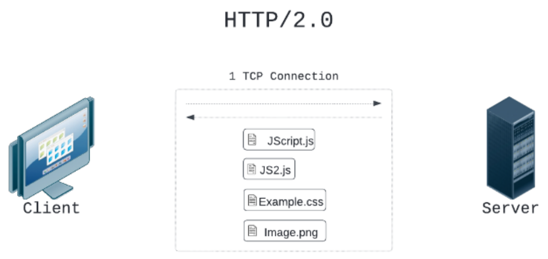
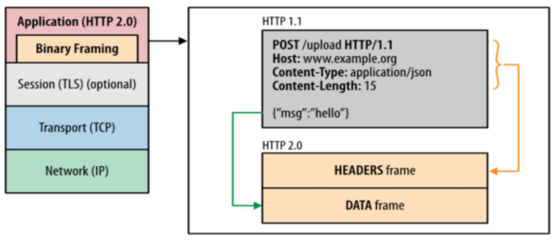


Figure 4: HTTP/2.0 Connection

HTTP/2 uses a binary farming layer that allows for multiplexing of requests and responses:



```
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15

{"msg":"hello"}
```

Figure 5: HTTP/2 Binary Framing

The binary framing mechanism in HTTP/2 means that multiple requests and responses can be performed at the same time, avoiding head-of-line blocking and reducing latency (Rahman & Rahman, 2017). Header compression is also used to reduce overhead. Server push allows servers to send responses to clients before a request is even made, further improving performance (Pollard, 2015). Streams are used to carry bidirectional exchanges, with each stream containing frames of end-user data or control information. Overall, HTTP/2's new features work together to make the web faster and more efficient compared to HTTP/1.1 (Wang et al., 2019).

HTTP/2 is widely regarded as an improvement over its predecessor, HTTP/1.x, but it is not without its disadvantages. One major disadvantage of HTTP/2.0 is its complexity, which can make it more difficult to implement and troubleshoot than HTTP/1.x (Bianchi et al, 2015). Another disadvantage is that HTTP/2 requires SSL/TLS encryption, which can increase the overhead of establishing a connection and processing requests (Wang et al., 2019). Additionally, while HTTP/2's multiplexing capability allows for faster page load times, it can also lead to head-of-line blocking if one request takes longer to process than others (Nam et al., 2016). Therefore, because of such a need for an improvement in web performances, has resulted in the invention of HTTP/3 and QUIC

3. Design

HTTP/2 was designed to address many of the performance issues of HTTP/1.x, but it still relies on TCP as its underlying transport protocol. TCP was designed for wired networks and is not optimized for high packet loss and delay that occur on wireless networks (Nam et al., 2016). QUIC was developed by Google (Iyengar & Swett, 2021), which is designed to reduce latency and improve performance over unreliable networks. HTTP/3, the latest version of the HTTP protocol, is designed to work with QUIC as its underlying transport protocol and build on the improvements of HTTP/2 while taking advantage of the benefits of QUIC such as reduced latency and improved performance over unreliable network.

The QUIC header contains several fields that are used to route and process packet, as well as provide security and reliability (Iyengar & Swett, 2021), the QUIC header includes the following fields:

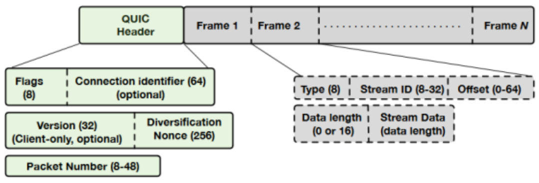

QUIC Header
Frame 1
Frame 2
Frame N
Flags (8)
Connection identifier (64) (optional)
Version (32) (Client-only, optional)
Diversification Nonce (256)
Packet Number (8-48)
Type (8)
Stream ID (8-32)
Offset (0-64)
Data length (0 or 16)
Stream Data (data length)

Figure 6: QUIC header

1. Flag: A set of flags that indicate various properties of the packet, such as whether it is a retransmission, or does it contain padding or not.

2. Connection Identifier: A unique identifier for the connection that the packet belongs to.

3. Version: The version of the QUIC protocol being used.

4. Diversification Nonce (D-Nonce): A randomly generated value that is used to protect against relay attacks.

5. Packet Number: A sequence number for the packet, which is used to detect lost or reordered packet.

A QUIC packet is divided into multiple frames, each of which contains a specific piece of data or control information:

1. Frame Type: Indicates the type of a frame, such as data frame, stream frame, or a connection close frame.

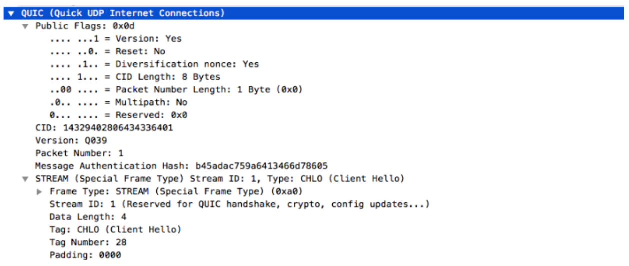
2. Stream ID: A unique identifier for the stream that the frame belongs to.

3. Offset: The byte offset within the stream where the data in the frame begins.

4. Data Length: The length of the data in the frame.

5. Data: The actual data being sent in the frame.

An example of how these binary bits work to form the QUIC packet is as follows:



```
▼ QUIC (Quick UDP Internet Connections)
  ▼ Public Flags: 0x0d
      .... ...1 = Version: Yes
      .... ..0. = Reset: No
      .... .1.. = Diversification nonce: Yes
      .... 1... = CID Length: 8 Bytes
      ..00 .... = Packet Number Length: 1 Byte (0x0)
      .0.. .... = Multipath: No
      0... .... = Reserved: 0x0
    CID: 14329402806434336401
    Version: Q039
    Packet Number: 1
    Message Authentication Hash: b45adac759a6413466d78605
  ▼ STREAM (Special Frame Type) Stream ID: 1, Type: CHLO (Client Hello)
    ▶ Frame Type: STREAM (Special Frame Type) (0xa0)
      Stream ID: 1 (Reserved for QUIC handshake, crypto, config updates...)
      Data Length: 4
      Tag: CHLO (Client Hello)
      Tag Number: 28
      Padding: 0000
```

Figure 7: QUIC packet

Together, these components combine and provide the groundwork for QUIC connection to be established.

4. Connection Establishment

A majority of services nowadays require secure and reliable network connection, TCP combining with TLS and QUIC are all transport protocols that are used to ensure secured connections between client and servers (Yong, Li, & Cong, 2012). The following figures will illustrate the difference and benefit of using UDP over TCP and TLS:

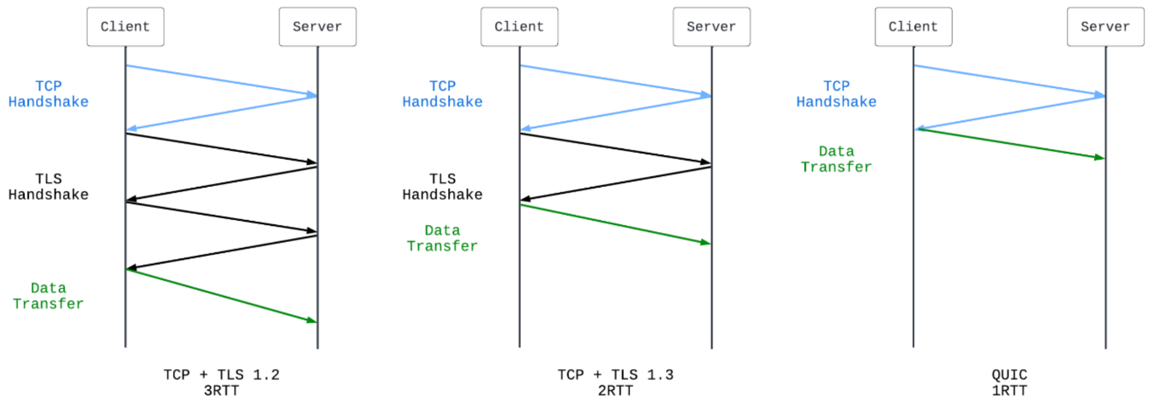


Figure 8: First Connection Establishment

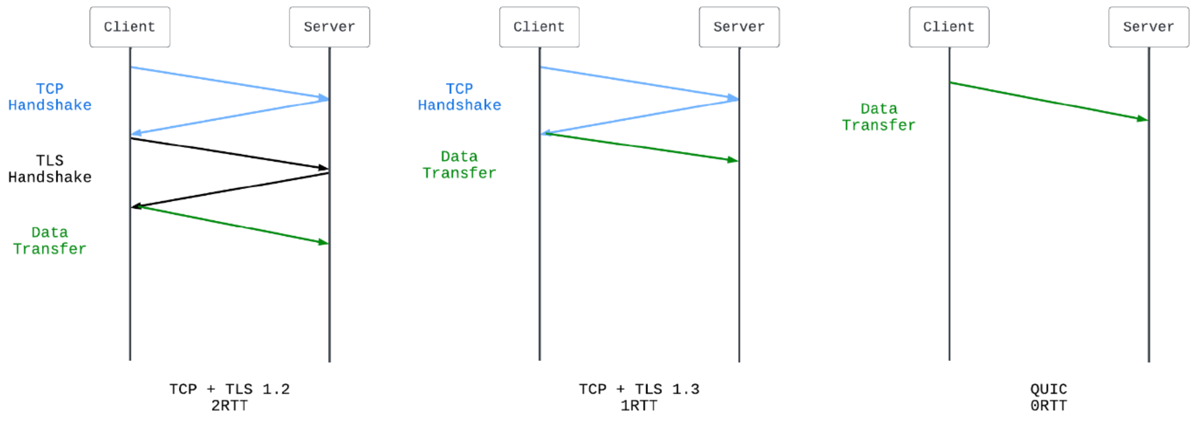


Figure 9: Subsequent Connection Establishment

First-time connection using TCP and TLS 1.2, apart from the well-known TCP three-way handshake, the client and server also engage in a process known as the "TLS handshake" (Dierks & Rescorla, 2008). During this process, the client and server exchange messages to authenticate each other. TCP and TLS 1.3 are similar to that of TLS 1.2; however, TLS 1.3 uses a reduced number of roundtrips and allows for faster key exchange. Although there has been improvement from TLS 1.2 to TLS 1.3, QUIC uses only one Round-Trip Time (RTT) for the first-time connection by combining the transport and crypto handshake, making it faster and more efficient than TCP with TLS 1.2 or TLS 1.3 (Yong, Li, & Cong, 2012).

Many connections between clients and servers which had communicated before can be established faster if the server could recognize the client during subsequent connections. TCP with TLS 1.2 requires at least 2RTT including 1 round-trip of TCP and 1 round-trip of TLS before the session can begin to exchange data. TCP with TLS 1.3 is faster with just 1RTT. However, as QUIC first connection has included encryption cookies stored on the client, future connection between the client with the same server can be done with 0RTT (Yong, Li, & Cong, 2012).

5. Benefits

Apart from the reduced latency offers by 0RTT connection establishment mechanism mentioned above, QUIC also comes with many other benefits that emerge as absolutely critical in today's web services.

5.1 Multiplexing

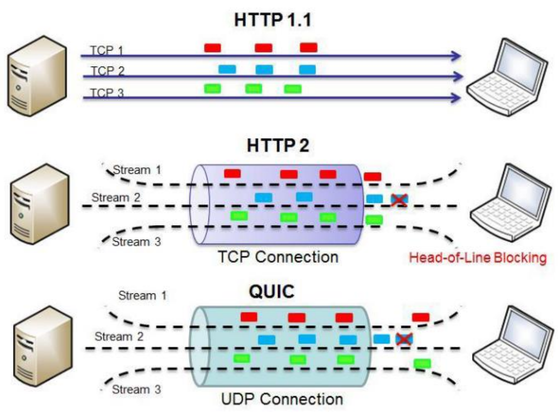


Figure 10: Multiplexing Comparison [innovating]

Multiplexing is a method for sending multiple streams of data over a single transport connection (Yong, Li, & Cong, 2012).

HTTP/1.1 must open multiple concurrent TCP connections to enable multiple requests to be sent over a connection, although the "keep alive" mechanism is introduced in this version, connection has to be serialized, nonetheless. This proves to be inefficient and was later improved by HTTP/2's ability to offer multiplexing, which allows multiple requests and responses to be processed in parallel. However, head-of-line blocking, which occurs when a slow or block request/response prevents other request/response from being exchanged, and therefore invites latency.

HTTP/3 using QUIC supports multiplexing of concurrent HTTP streams without requiring ordered delivery of all packets of the transport connection, and therefore responses can be processed simultaneously without being blocked or slow or lost packets (Iyengar & Sweet, 2021).

5.2 Congestion Control

QUIC provides a slightly different approach for congestion control than TCP. QUIC adopts modern loss recovery mechanisms such as F-RTO (Henderson et al, 2012) and Cubic (Allman M., et al, 2010). It is designed to be more adaptive and responsive than TCP's congestion control. Overall, these techniques help to ensure that QUIC can adapt to changing network conditions and maintain a high level of performance while avoiding congestion.

5.3 Loss Recovery

QUIC includes a loss recovery mechanism that is designed to be more efficient and reliable than TCP's. Time-Based Acknowledgements and Tail Loss Probes are the most noteworthy of them all (Iyengar & Swett, 2019). Time-Based Acknowledgement allows QUIC to provide more accurate feedback on the state of the network, this may include the arrival time of packets, which helps to ensure that lost packets are detected and retransmitted more quickly (Iyengar & Swett, 2019). Tail Loss Probes, on the other hand, also ensures that lost packets are detected and retransmitted as quickly as possible. This is done by sending extra packets at the end of transmission to determine if any packet was lost (IETF, 2018).

6. Drawbacks

While QUIC offers several benefits for web communication, it also has some potential disadvantages and limitations.

6.1 Increased Complexity

QUIC is a complex protocol that includes many advanced features and mechanisms (Iyengar & Swett, 2019). This can make it more difficult to implement and debug, particularly for developers and system administrators who are not familiar with the protocol.

6.2 Resource Consumption

QUIC can consume more resources on the client and server, including memory, processing power, and network bandwidth. This can be a concern for systems with limited resources, or in situations where many clients are connected to a single server.

7. Conclusion

In conclusion, QUIC is a promising new protocol for web communication that offers several benefits over traditional TCP-based protocols. Its motivation stems from the need for a more efficient and reliable transport protocol that can address the limitations of TCP in high-latency or lossy network environments.

Overall, QUIC represents a significant step forward in the evolution of web communication, and its adoption is likely to increase in the coming years as more web browsers and servers begin to support the protocol. However, it is important to consider the potential disadvantages and limitations of QUIC and further research is needed to explore the full potential of QUIC and its implications for web communication. Nevertheless, QUIC is a promising technology that has the potential to significantly improve the user experience and optimize the use of network resources for web applications.

8. References

[1]. Yong, C., Tianxiang, L., Cong, L. (2012). Innovating transport with QUIC: Design approaches and research challenges. IEEE, 21(2), 72-76.

[2]. Puneet, K. (2020). QUIC - A Quick Study. Santa Vlara University. Retrieved from: https://www.researchgate.net/publication/344490561_QUIC_Quick_UDP_Internet_Connections

[3]. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1996). Hypertext Transfer Protocol -- HTTP/1.0. Retrieved from https://www.w3.org/Protocols/HTTP/1.0/spec.html

[4]. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1997). Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068. Retrieved from https://tools.ietf.org/html/rfc2068

[5]. Rahman, M. S., & Rahman, M. S. (2017). A Comparative Study of HTTP/1.1 and HTTP/2 Protocols. International Journal of Computer Science and Network Security, 17(1), 39-47.

[6]. Zang, Y., & Zhu, Y. (2018). Performance evaluation of HTTP/2 in modern Web and mobile Devices. In Proceedings of the 2018 IEEE International Conference on Communications (ICC) (pp. 1-6). IEEE.

[7]. Pollard, B. (2015). HTTP/2. O'Reilly Media.

[8]. Bianchi, G., Rossi, D., & Vitucci, F. (2015). Analysis of QUIC Performance on Mobile Networks. In Proceedings of the 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 86-91). IEEE.

[9]. Wang, X., Cui, L., Yang, Y., & Zhang, Y. (2019). Performance Evaluation of HTTP/2 and QUIC Protocols in Edge Computing. In Proceedings of the 2019 5th IEEE International Conference on Computer and Communications (ICCC) (pp. 1885-1889). IEEE.

[10]. Nam, Y., Lee, J., Lee, J., & Choi, Y. (2016). Performance Analysis of HTTP/2 Server Push for Adaptive Streaming in Mobile Networks. In Proceedings of the 2016 IEEE International

Conference on Consumer Electronics (ICCE) (pp. 194-195). IEEE.

[11]. Iyengar, J., & Swett, I. (2020). QUIC: Quick UDP Internet Connections. Retrieved from: https://www.researchgate.net/publication/344490561_QUIC_Quick_UDP_Internet_Connections

[12]. Dierks, T., & Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. https://tools.ietf.org/html/rfc5246

[13]. Henderson, T. R., Floyd, S., Gurtov, A., & Nishida, Y. (2012). The F-RTO Algorithm for Detecting Spurious Retransmissions. RFC 5682. https://tools.ietf.org/html/rfc5682

[14]. Allman, M., Ostermann, S., Wang, C., & Blanton, E. (2010). TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824. https://tools.ietf.org/html/rfc6824

[15]. Iyengar, J., & Swett, I. (2019). QUIC: A UDP-Based Multiplexed and Secure Transport. IETF Internet-Draft. https://tools.ietf.org/html/draft-ietf-quic-transport-28