COMP5311 Project

Topic: Comparison of TCP and QUIC

GroupID: 23

20002294G Markus Tse

20004702G Chan Ho Ken

21000018G Li Yat Long

21009757G Lee Ka Chun Jacky

Contribution Table:

TCP Sever Program	20002294G Markus Tse
QUIC Server Program	21009757G Lee Ka Chun Jacky
Test Cases	20002294G Markus Tse 21000018G Li Yat Long
Testing	21009757G Lee Ka Chun Jacky
Report (Abstract)	20004702G Chan Ho Ken
Report (Protocol explanation: TCP, UDP, QUIC, TLS)	20002294G Markus Tse 20004702G Chan Ho Ken 21000018G Li Yat Long
Report (Methodology, Test Case, Clumsy, Server Implementation)	20004702G Chan Ho Ken 21000018G Li Yat Long
Report (Results)	21000018G Li Yat Long 21009757G Lee Ka Chun Jacky
Report (Discussion)	21000018G Li Yat Long 21009757G Lee Ka Chun Jacky
Report (Conclusion)	20002294G Markus Tse 21000018G Li Yat Long
Presentation Slides	All members

1. Abstract

In computer networking, the transport layer provides host to host communication for different applications, each protocol has different advantages and disadvantages. Every couple of years, a new protocol will be developed to target existing issues, may it be communication time, reliability or security issues. This paper will introduce two transport layers — Transmission Control Protocol, TCP, and Quick UDP Internet Connection, QUIC.

TCP originates from an initial network implementation that complements Internet Protocol (IP). It is one of the common network protocols we used in the network. It was introduced in the 1970s and provides reliable data transfer .

QUIC (Quick UDP Internet Communication) proposed by Google, is a new multiplexed transport protocol built on top of UDP. It aims at improving performance of connection-oriented web applications that are currently using TCP. Various Google owned websites like YouTube, Google Search are using QUIC.

This project is aimed to analyse if QUIC is superior to TCP by comparing the performance in packet transmission speed and why TCP is still the most popular transport layer protocol.

2. Transport Layer Protocol

Transport Layer Protocol is one of the layers in the network OSI model. It provides services such as reliability, connection-oriented communication, flow control and multiplexing. It can be represented by two typical examples: TCP and UDP.

2.1 TCP

The origin of TCP, which was proposed in 1974 to support reliable data transfer. In order to establish a reliable connection, TCP introduced the three way handshake between client and server state.

2.1.1 Handshake

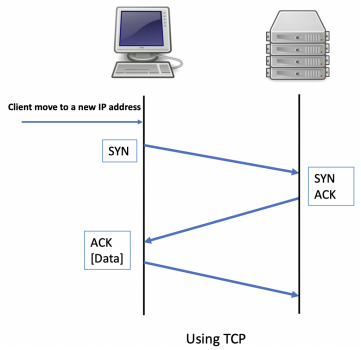

Fig. 1. TCP handshake by Dellaverson et al. [1]

The client will send a TCP SYN message to the server with a sequence number specified. After the server receives the message, it will reply with an ACK message with the next number of the sequence number received and SYN with the a sequence number specified. Finally, if the client received the message from the server, it acknowledges the server is live and can send back the ACK with the next number of the sequence number received to let the server acknowledge the client is live.

2.1.2 Header

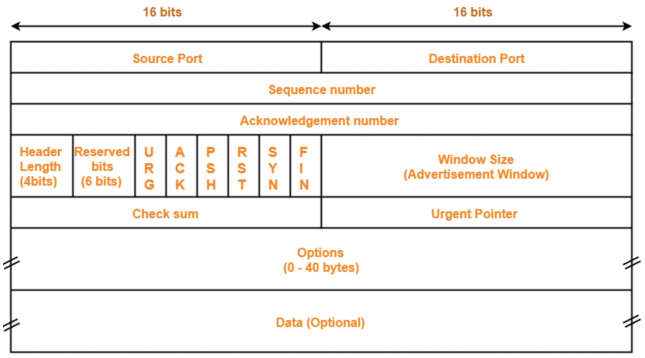

Fig. 2. TCP Header format by Singhal [2]

The above Fig. 2 shows the design of TCP header with a total 20 bytes of data stored not counting the optional parts. The header mainly contains the sequence number of the packet, ACK, connection (SYN/ FIN) message, checksum of the packet and the window size which represent the amounts of data the sender can send.

2.1.3 Congestion control

In TCP Reno, three modes of TCP's congestion control are designed to ensure the client and network can handle the amount of data sent from the sender, which includes slow-start, congestion avoidance and fast recovery.

2.1.3.1 Slow start

The sender using TCP will enter a slow start in the beginning with one segment per transmission and the number of segments allowed will increase twice every time in the next round trip until the number of segments allowed exceeds the slow start threshold.

2.1.3.2 Congestion Avoidance

Once the number of segments allowed to be transmitted exceeds the slow start threshold in slow start mode, the congestion control of TCP will be transferred to congestion avoidance which increases the number of segments allowed by one every round trip.

2.1.3.3 Fast Recovery

If the sender receives 3 duplicate ACK which represent packet loss during the transmission, the number of segments allowed to be transmitted will be halved, the lost packet will be retransmitted and fast recovery congestion control will be activated from slow start/ congestion avoidance. During fast recovery, if the sender received ACK, it will roll back to congestion avoidance state. Otherwise, it will experience a timeout and roll back to a slow start state.

2.2 UDP

User Datagram Protocol (UDP) is a transport protocol used to establish low-latency and loss-tolerating connections between applications on the internet. Both UDP and TCP run on top of IP and are sometimes referred to as UDP/IP or TCP/IP. However, compared to TCP, there is no handshaking on UDP to ensure reliable data transfer, resulting in packet loss or out of order occur easily on UDP.

2.3 TLS

Transport Layer Security (TLS) is a widely used security protocol designed to protect data for internet communications between different network hosts and servers. There are three main components of TLS accomplishes: Encryption, Authentication, and Integrity. A TLS

certificate will be installed on the host server, the certificate contains a lot of information, one of them is the server's public key. When a client connects to the server a TLS handshake will be performed and it establishes a cipher suite for this communication session. The cipher suite is a set of algorithms that specifies details such as which shared encryption keys, or session keys, will be used for that particular session [3].

2.4 QUIC

QUIC proposed by Google is the new alternative to the traditional HTTP/2+TCP+TLS combination, replacing TCP as the transport layer protocol. Google [4] claims to reduce connection establishment time, congestion control feedback, multiplexing, connection migration, transport extensibility and optional unreliable delivery.

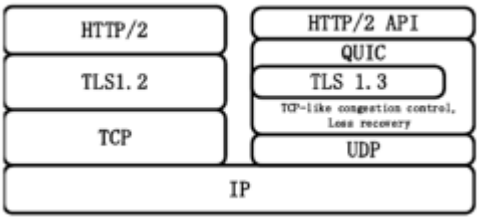

Fig. 3. Traditional HTTP+TLS+TCP architecture vs QUIC architecture by [5]

The above Fig. 3 shows that the architecture of QUIC is not completely new, which is based on another existing transport layer protocol - UDP. Iyengar and Thomson [6] points out that the design of creating a new transport layer protocol based on the existing UDP for transfering is to ensure the new protocol can be implemented easily on existing systems and networks. Meanwhile, Compared to TCP which rely on additional protocols like TLS for encryption by sending additional TLS messages between client and server, QUIC combines the handshake of TCP and cryptographic to one message, reducing the RTT significantly from three to one [5]. Ideally, the RTT can be further reduced to 0RTT by saving the host name and the port number of the previously visited server, resulting in direct communication to the server without additional handshake [5].

2.4.1 Header

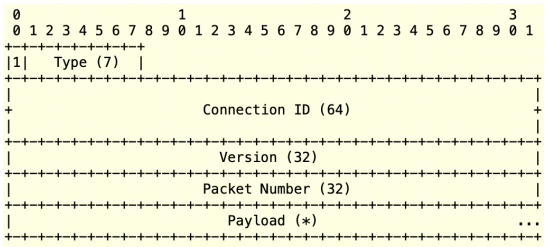

Fig. 4. QUIC Long Header format by Iyengar and Thomson [7]

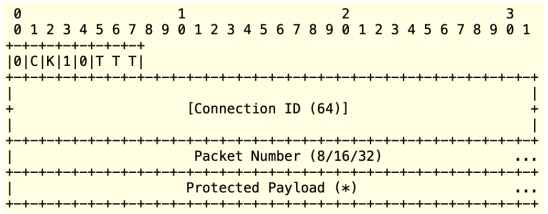

Fig. 5. QUIC Short Header format by Iyengar and Thomson [7]

The above Fig. 4 and Fig. 5 demonstrate the two header formats of QUIC. Unlike TCP, QUIC comes with both long header format and short header format, with the main difference between the long and short header format is the absence of version number in short header version and the packet number can be reduced to 8 and 16 bits from 32 bits. The longer header is only used if the version negotiation and the establishment of 1-RTT keys is not completed. Otherwise, the shorter header will be applied to all cases.

Both long and short header stores the information of connection ID and packet number. According to Iyengar and Thomson [7], the packet number of a QUIC header is to identify each cryptographic nonce for packet encryption which the number cannot be reused by another packet within the same connection. On the other hand, Iyengar and Thomson [7] also explained that the 64 bit connection ID is used for identifying the QUIC connection, which is initially chosen by the client during the initial connection and re-selected by the server-side after the server receives the initial connection request.

2.4.2 Congestion control

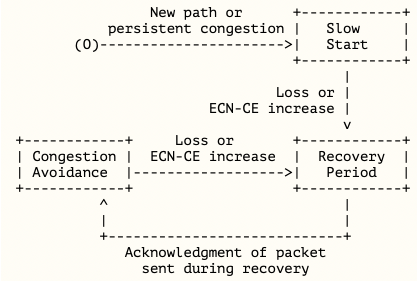

Fig. 6. QUIC congestion control states and transitions by Iyengar and Swett [8]

The above Fig 6 demonstrates that one kind of congestion control of QUIC is extremely similar to congestion control of TCP NewReno which is an updated version of RENO mentioned before, achieving multiple packet loss. The sender will begin in slow start state until the congestion window is larger or equal to the slow start threshold or loss detected, where state will be transferred to congestion avoidance or recovery. Meanwhile, Iyengar and Swett [8] also mentioned that QUIC can detect the loss of packets only containing ACK frames and use that information to adjust the congestion controller or the rate of ACK-only packets being sent, where TCP cannot.

In our project, the slow start of QUIC program is modified using Hybrid Slow Start, which offers trust-worthy signals to switch from slow start to congestion avoidance safely, while maintaining the mechanism of slow start in TCP NewReno and preventing a large number of packet loss [9].

3. Methodology

In our test, Image files were transmitted from the TCP/ QUIC server-side to the client side with or without network latency and packet loss. Each test case was tested five times to ensure the results accuracy. The below PC were used as our test environment:

- OS: 64-bit Windows 10
- Hardware: Intel Core i7-6700HQ CPU @ 2.60GHz (with SSE4.2) and 8 GB of RAM

3.1 Test case

Fig. 7. Test Image for our test

The above Fig. 7 were prepared in 4 sizes including 12KB, 352KB, 1.1MB, 3MB, covering the common file sizes found during Internet browsing. Meanwhile, Network latency or packet loss including no latency, 20ms and 100ms latency (sum of inbound and outbound latency), no packet loss, 1% and 5% packet loss were applied on both TCP and QUIC server.

Case	Image Size	Network Latency	Packet Loss
1	12KB	0ms	0%
	352KB		
	1.1MB		
	3MB		
2	12KB	20ms	0%
	352KB		
	1.1MB		
	3MB		
3	12KB	100ms	0%
	352KB		
	1.1MB		

	3MB		
4	12KB	0ms	1%
	352KB		
	1.1MB		
	3MB		
5	12KB	0ms	5%
	352KB		
	1.1MB		
	3MB		

Fig. 8. All test cases of our testing

3.2 Clumsy

Clumsy, which is an open source Windows program simulating network latency, packet discard, throttling, duplicate and out order packet on internet and localhost connection [10], is used to configure the inbound and the outbound connection in order to simulate packet loss and latency.

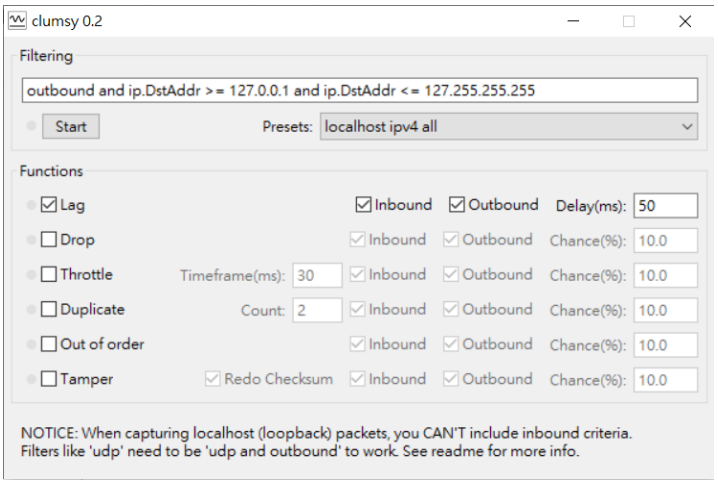

Fig. 9. Clumsy setting screen

3.3 TCP Server Implementation

A Python program with flask was implemented for hosting a simple TCP server with few APIs for requesting the image files from the client-side. Shukla [11] explanies that Flask which is based on WSGI (Web Server Gateway Interface) toolkit and Jinja2 template engine, is an easily implemented Python API for building simple web-application.

Once the localhost server had been activated, Postman, which is a platform for implementing API testing easily, was used for obtaining the images from the server-side by HTTP GET method. Meanwhile, Wireshark, which is a network protocol analyser supporting packet capture on wired, wireless and localhost networks, was used for measuring the time required of file transmission in each test case.

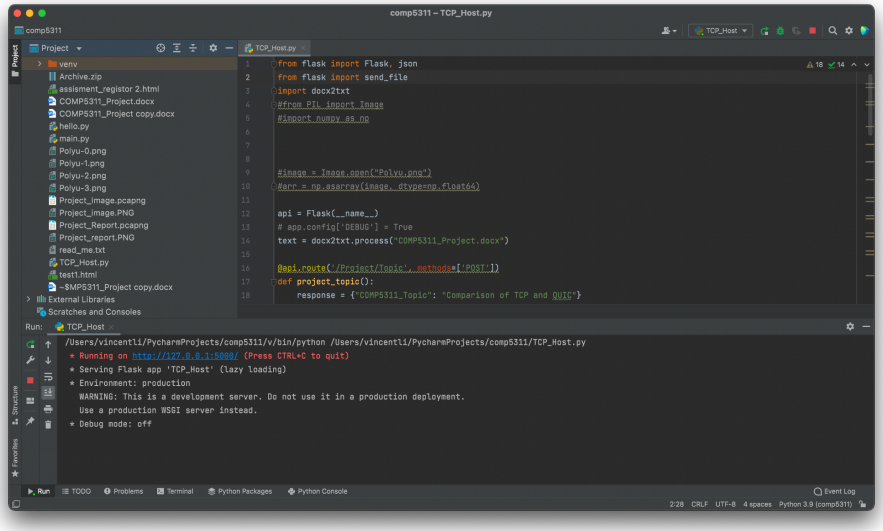

Fig. 10. Python program for TCP server hosting

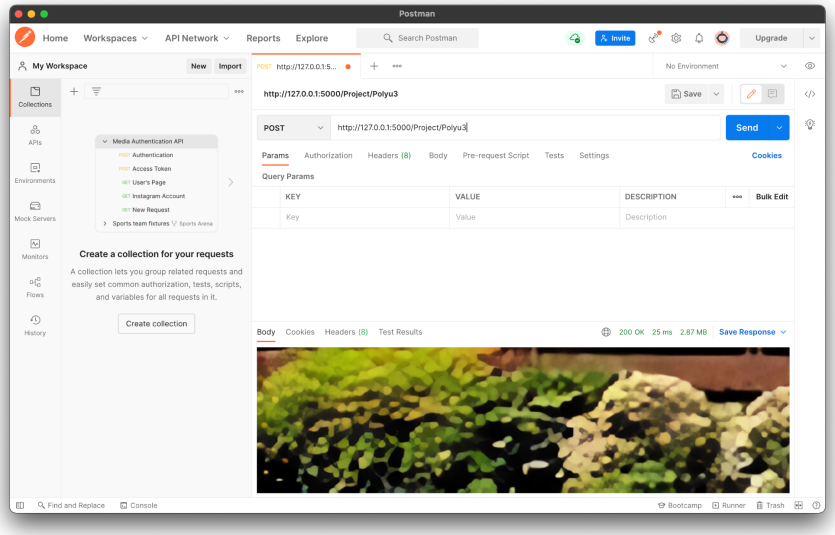

Fig. 11. Using Postman to perform HTTP GET

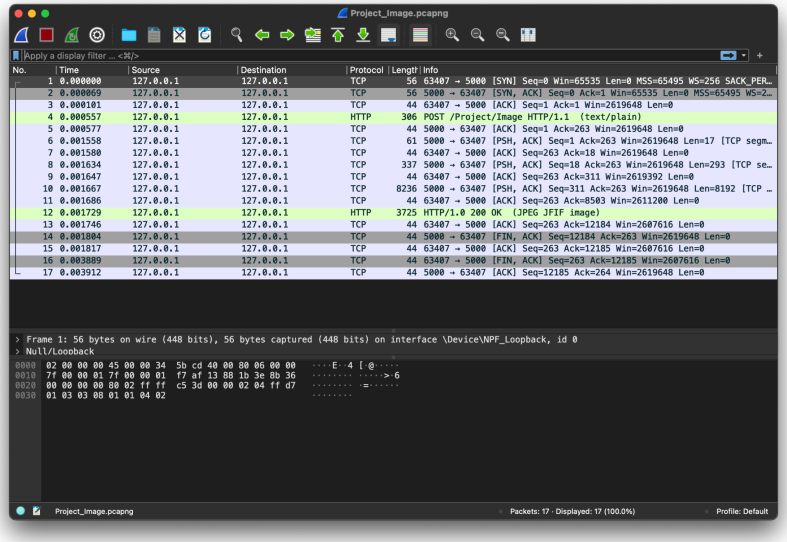

Fig. 12. Using WireShark to capture packet transmission

3.4 QUIC Server and Client Implementation

Go-libp2p-quic-transport [12] was used to build up the quic server and client. As the initial code could only transmit text messages, some modifications were done in the go files so that images could be transmitted and encoded into base64. The files were executed through Powershell with the following procedures:

First, the go program on the server side was executed and a peer id was generated. Then, the go program of the client side was run with the peer id and a message would be sent to the server. When the server received the message, it would read the png file and sent it to the client. The client would encode the data sent from the server to base64 so that we could check the png file. Meanwhile, Same as the method in testing the performance of TCP, we used Wireshark for obtaining the result of QUIC.

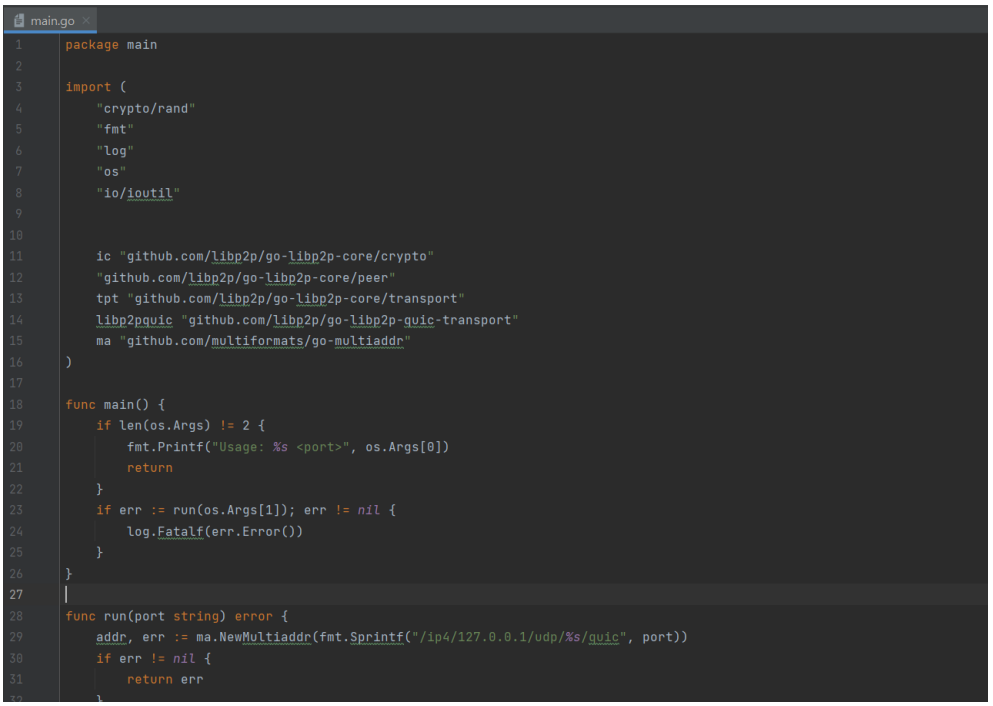

Fig. 13. Go program for QUIC server hosting

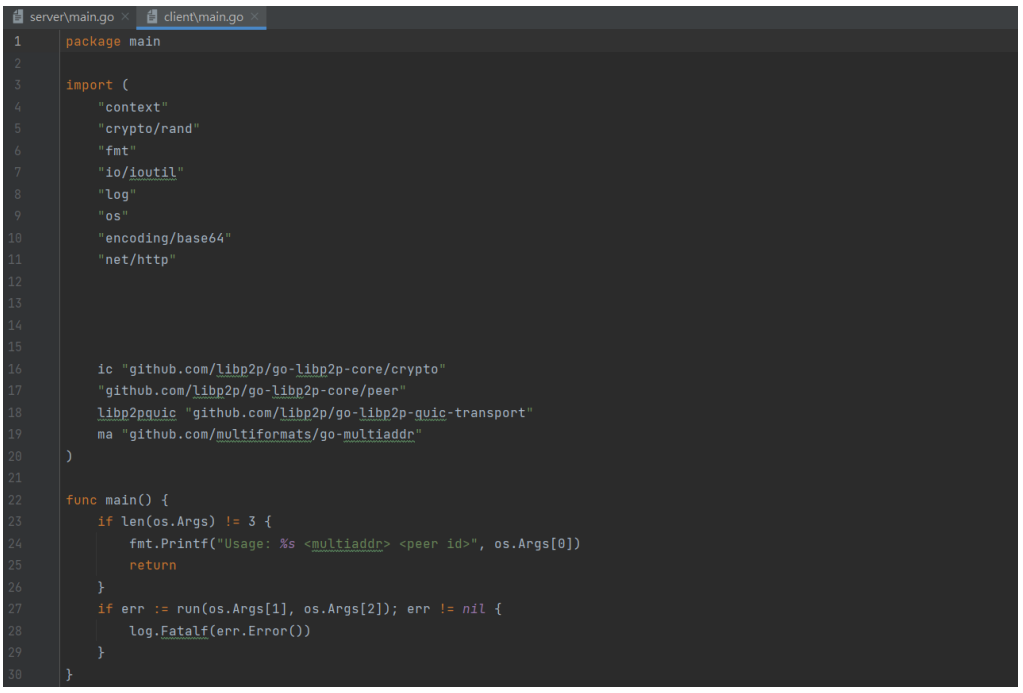

Fig. 14. Go program for QUIC client hosting

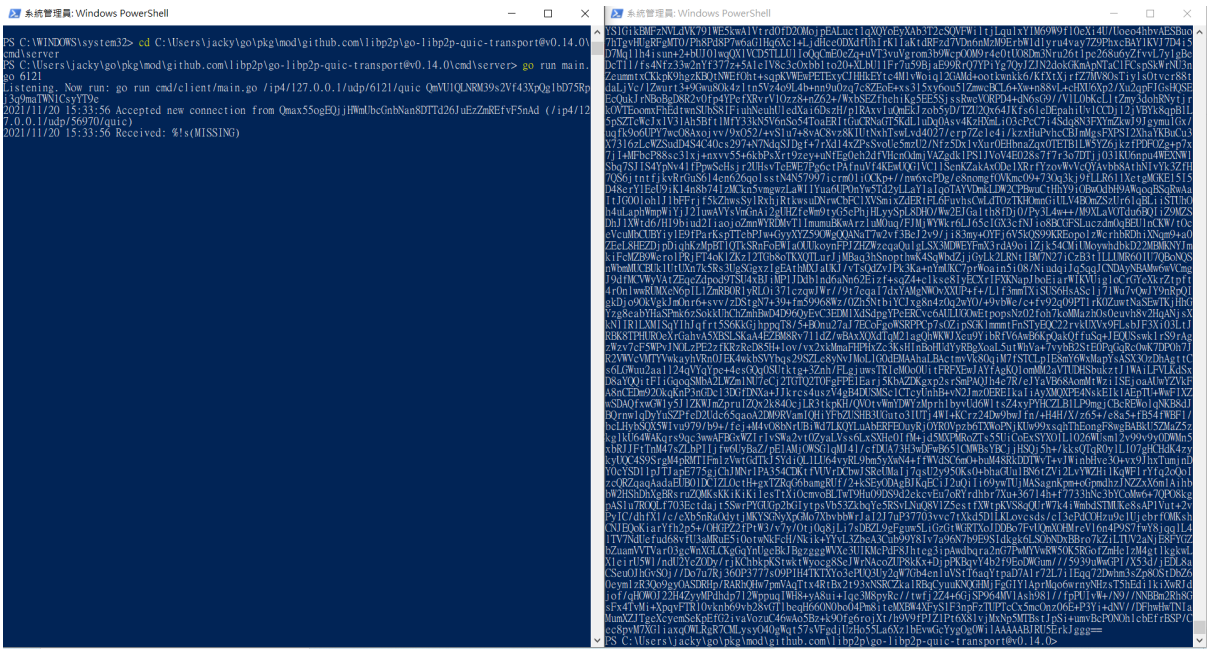

Fig. 15. Using Powershell to execute the go files

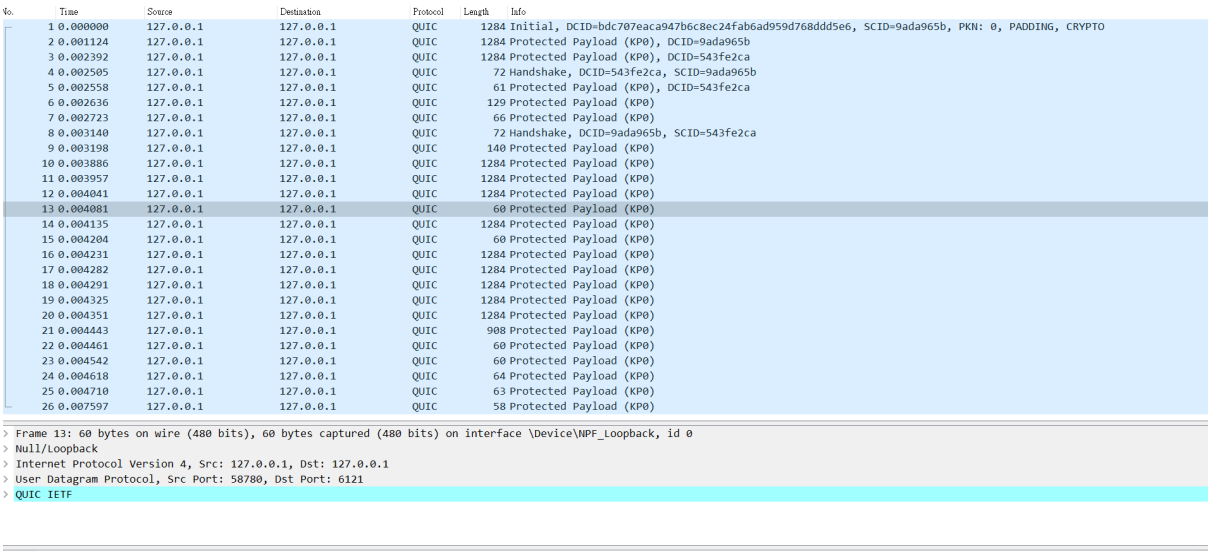

Fig. 16. Using WireShark to capture QUIC packet transmission

4. Results

Test Case 1	TCP time (average) (second)	TCP time (min,max) (second)	QUIC time (average) (second)	QUIC time (min,max) (second)
Image 12KB	0.01101725	[0.009338, 0.016584]	0.0080628	[0.006710, 0.018354]
Image 352KB	0.01807575	[0.014062, 0.023764]	0.07988675	[0.073717, 0.090452]
Image 1.1MB	0.025867666	[0.022638, 0.029278]	0.2418905	[0.208639, 0.292481]
Image 3MB	0.05134975	[0.043448, 0.056726]	0.629232	[0.592975, 0.689222]

Fig. 17. The test results of Test Case 1 (no latency and packet loss)

The above Fig. 17 shows the results when there was no network latency and packet loss. QUIC only performed better if the Image size was 12KB with average 26.82% faster. In the

test with other file sizes, QUIC performed much worse than TCP with average 341.96% slower when the image is 352KB to average 1125.38% slower when the image is 3MB.

Test Case 2	TCP time (average) (second)	TCP time (min,max) (second)	QUIC time (average) (second)	QUIC time (min,max) (second)
Image 12KB	0.214122	[0.168734, 0.257944]	0.234048	[0.2114, 0.307929]
Image 352KB	0.575984	[0.46101, 0.84602]	0.810689	[0.80671, 0.816802]
Image 1.1MB	1.214376	[1.038068, 1.484907]	2.246472	[2.214569, 2.288493]
Image 3MB	3.025806	[2.383631, 3.355772]	5.969542	[5.941724, 5.986712]

Fig. 18. The test results of Test Case 2 (20ms latency)

The above Fig. 18 shows the results when there was 20ms latency. On average, TCP out performed QUIC with maximum 49.31% faster when image size was 3MB. However QUIC demonstrated more stable performance than TCP when image size was 352KB, 1.1MB and 3MB with only 1.25%, 3.34% and 0.76% difference respectively.

Test Case 3	TCP time (average) (second)	TCP time (min,max) (second)	QUIC time (average) (second)	QUIC time (min,max) (second)
Image 12KB	0.410187	[0.405204, 0.410703]	0.3569484	[0.310485, 0.488844]
Image 352KB	1.573486	[1.545928, 1.622420]	0.8688894	[0.843946, 0.902042]
Image 1.1MB	4.327565	[4.311464, 4.356912]	1.5304716	[1.357436, 1.688627]
Image 3MB	11.4678466	[11.455842, 11.478721]	3.5083998	[3.095998, 3.719689]

Fig. 19. The test results of Test Case 3 (100ms latency)

The above Fig. 19 shows the results when there was 100ms network latency. QUIC performed better than TCP in all transmissions with average 12.9% faster when the image is 12KB to average 69.41% faster when the image is 3MB.

Test Case 4	TCP time (average) (second)	TCP time (min,max) (second)	QUIC time (average) (second)	QUIC time (min,max) (second)
Image 12KB	0.0104382	[0.006736, 0.015043]	0.007422	[0.006656, 0.00854]
Image 352KB	0.115182667	[0.010525, 0.32251]	0.094731	[0.091357, 0.100267]
Image 1.1MB	0.1527	[0.021322, 0.347858]	0.262037	[0.252651, 0.273018]
Image 3MB	1.033641	[0.045795, 2.817762]	0.655404	[0.639719, 0.667677]

Fig. 20. The test results of Test Case 3 (1% packet loss)

The above Fig. 20 shows the results when there was 1% packet loss. QUIC showed its advantages when the image size was 12KB with average 28.9% faster and 352KB with average 17.76% faster. Meanwhile, TCP performed better and worse at the same time when the image size was 1.1MB and 3MB. When the image size was 1.1MB, the best case of TCP performed 91.56% faster than the best case of QUIC. However, the worst case of TCP with the same image size performed 27.41% slower than the worst case of QUIC with the same image size. The above scenario also occurred similarly when Image size was 3MB, the transmission time of TCP was not consistent.

Test Case 5	TCP time (average) (second)	TCP time (min,max) (second)	QUIC time (average) (second)	QUIC time (min,max) (second)
Image 12KB	0.0798998	[0.007942, 0.341037]	0.016966	[0.007574, 0.041013]
Image 352KB	0.7736284	[0.035008, 1.646417]	0.104461	[0.094848, 0.120027]
Image 1.1MB	1.8299898	[0.647229, 4.721869]	0.265376	[0.239803, 0.295846]
Image 3MB	3.0824064	[2.204153, 3.743857]	0.829224	[0.762649, 0.910647]

Fig. 21. The test results of Test Case 3 (5% packet loss)

The above Fig. 21 shows the results when there was 5% packet loss. QUIC performed better in all file sizes with more stable results compared to TCP, except the best case of TCP with file size of 352KB which TCP performed 63.09% better than QUIC's best case when transferring the same file size. The instability of TCP can be shown when transfering file size of 1.1MB, which in the best case is 86.29% faster than the worst case. Compared to QUIC transferring the same 1.1MB file, the best case is 18.94% faster than the worst case, representing it's much more stable performance compared to TCP.

All in all, QUIC performed faster than TCP in most cases when there network latency and offered more stable performance when there were packet loss. However, if there were no network latency and packet loss, QUIC only showed its advantages if the image size was 12KB.

5. Discussion

In the case of no latency and packet loss, QUIC outperformes TCP only when the file size is 12KB. However, when the file becomes larger, TCP has better performance. QUIC' s

performance gain for smaller object sizes is mainly due to QUIC's 0-RTT connection establishment [13]. For larger object sizes, this effect becomes not significant to the loading time. Also, TCP works very well for bulk transfers when latency or packet loss is low or zero. Hence, the advantages of QUIC in congestion control cannot be shown when the connection is extremely good.

In the case with extra loss or latency, QUIC outperforms TCP due to its stream multiplexing. When one stream is blocked due to a missing packet, QUIC still has other streams to transmit the data [13]. However, if the loss occurs in TCP, all streams need to wait for packet recovery. Also, the congestion algorithm of QUIC is better so it can estimate the RTTs and detect and recover the loss more quickly.

On the other hand, some of our test results match the results done by other research.

Parameter	Values Tested
Bandwidth	10mbps
Extra loss	0%, 0.1%, 1%
Extra delay (RTT)	50ms, 100ms
Single-object sizes	100KB, 1MB, 5MB
Web-page sizes	small (≤ 0.47MB) medium (≤ 1.54MB) large (≤ 2.83MB)
Content Providers	Facebook, Google, Cloudflare

Fig. 22. QUIC and TCP test scenario by Yu and Benson [14]

The above Fig 22 shows the test cases chosen by Yu and Benson on setting up their test by using QUIC and TCP against public endpoints from Google, Facebook and Cloudflare by transmissing static files and webpages with loss and latency similar to our test. According to their test results [14], by sending a single-object, QUIC only outperformed TCP when there were extra delays on google server or the payload is smaller than 1MB if there were latency through google or facebook server or without loss and latency. Meanwhile, when transmissing multiple objects, QUIC performed better than TCP for small sized web pages when there was latency through Google and Cloudflare server [14]. This test matches our results with a small file size where QUIC usually performed better than TCP. However, unlike our test results, their results did not see advantages of QUIC when the payload was

large and there was latency or loss, which could be explained by the different congestion control implemented by different servers [14].

Meanwhile, other studies also demonstrated that QUIC is not perfect, in many cases TCP still performed better. An experiment done by Saif et al. [15] shows that QUIC performed significantly worse than TCP on largest contentful paint which measures the time needed on the page with largest payload to be fully rendered. In other metrics like first contentful paint, time to interactive and speed index, QUIC benefits were not apparent and in many cases TCP were still better [15]. On the other hand, Nepomuceno et al. [16] compared QUIC and TCP by utilized them to access hundred most accessed pages worldwide with the results of every config of TCP performed better than QUIC on at least 60% of pages visited and enabling cache on TCP produced higher performance improvement than using QUIC.

On the other hand, Yu and Benson [14] also mentioned that there are more factors affecting the results besides the design of QUIC, especially on large payloads, including but not limited to congestion control and TLS configuration on the client side not matching the config on the server side. Meanwhile,implementing QUIC to achieve good performance is not a simple task as optimising QUIC's congestion control is complex even for Facebook [14], providing another reason why TCP is still popular.

6. Conclusion

In this paper we have presented our analysis between QUIC and TCP and tested the performance difference between both protocols under a stable and unstable network. The result clearly shows that QUIC consistently outperforms TCP when more then 20ms latency or package loss is introduced into the network, however the performance is not obvious in a stable network the performance or in real world scenario. TCP clearly is unable to be replaced by QUIC for now by its small advantages in performance and its cost of implementation. In our future work we plan to further compare their difference by deploying host and client server in a different device and test the difference between LAN/WIFI and mobile network.

References

[1] J. Dellaverson, T. Li, Y. Wang, J. Iyengar, A. Afansyev and L. Zhang, "Understanding QUIC," presented at Conference'17. [Online]. Available: http://web.cs.ucla.edu/~lixia/papers/UnderstandQUIC.pdf [Accessed Nov 3 ,2021]

[2] A. Singhal, *TCP Header | TCP Header Format | TCP Flags,* 2020 [Online]. Available: https://www.gatevidyalay.com/transmission-control-protocol-tcp-header/ [Accessed Nov. 18, 2021]

[3] Cloudflare, *What is a cryptographic key? | Keys and SSL encryption,* 2021 [Online]. Available: https://www.cloudflare.com/en-gb/learning/ssl/what-is-a-cryptographic-key/ [Accessed Oct. 21, 2021]

[4] Google, *QUIC, a multiplexed transport over UDP*, n.d. [Online]. Available: https://www.chromium.org/quic[Accessed Oct. 7, 2021]

[5] J. Zhang et al., "Formal Analysis of QUIC Handshake Protocol Using Symbolic Model Checking," *IEEE Access*, vol. 9, pp. 14836-14848, Jan. 2021. [Online]. Available: https://www.researchgate.net/publication/348799241_Formal_Analysis_of_QUIC_Handshake_Protocol_Using_Symbolic_Model_Checking[Accessed Oct. 18, 2021]

[6] J. Iyengar, and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," *RFC9000*, May 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9000 [Accessed Oct. 7,2021]

[7] J. Iyengar, and M. Thomson, QUIC: A UDP-Based Multiplexed and Secure Transport. [Online]. Available: https://tools.ietf.org/id/draft-ietf-quic-transport-10.html#rfc.section.3.1 [Accessed Oct. 22, 2021]

[8] J. Iyengar, and I. Swett, "QUIC Loss Detection and Congestion Control," *RFC9002,* May 2021. [Online]. Available: https://datatracker.ietf.org/doc/rfc9002/ [Accessed Nov. 1,2021]

[9] S. Ha, and I. Rhee, “Hybrid Slow Start for High-Bandwidth and Long-Distance Network,” presented at International Workshop on Protocols for Fast Long-Distance Networks, 2008. [Online]. doi: 10.1145/1851182.1851192. [Accessed Nov. 10, 2021]

[10] Jagttt, *Clumsy 0.2*, n.d. [Online]. Available: https://jagt.github.io/clumsy/ [Accessed Nov. 13, 2021]

[11] K. Shukla “Python | Introduction to Web development using Flask,” 2020. [Online]. Available: https://www.geeksforgeeks.org/python-introduction-to-web-development-using-flask/ [Accessed Nov, 5 2021]

[12] M. Seemann, *libp2p / go-libp2p-quic-transport*, 2021. [Online]. Available: https://github.com/libp2p/go-libp2p-quic-transport/ [Accessed Oct 30, 2021]

[13] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, “Taking a Long Look at QUIC,” presented at IMC ’17. [Online]. Available: https://conferences.sigcomm.org/imc/2017/papers/imc17-final39.pdf?fbclid=IwAR3AMIGvgMnNbQeQdOdHjsBxN_3pGr-xTXIiM5DpEFAw7a7FgU0BgZAGCow [Accessed Nov 1, 2021]

[14] A. Yu, and T. A. Benson, “Dissecting Performance of Production QUIC,” presented at The Web Conference 2021. [Online]. Available: https://cs.brown.edu/~tab/papers/QUIC_WWW21.pdf [Accessed Oct. 20, 2021]

[15] D. Saif, C. H. Lung and A. Matrawy, “An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse,” Department of Systems and Computer Engineering, Carleton University, Ottawa, Oct. 2020. [Online]. Available: https://arxiv.org/pdf/2004.01978.pdf [Accessed Oct. 28, 2021]

[16] K. Nepomuceno et al., "QUIC and TCP: A Performance Evaluation," *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00045-00051, 2018. [Online] doi: 10.1109/ISCC.2018.8538687 [Accessed Nov 3, 2021]