International Journal of
Computer & Software Engineering

Original Article Open Access

Evaluation of a File Transfer Protocol with QUIC

Yudai Ishikawa and Tomofumi Matsuzawa*
Department of Information Sciences, Tokyo University of Science, Tokyo 162-8601, Japan

Abstract

In May 2021, a new transport layer protocol, QUIC, was standardized by the Internet Engineering Task Force (IETF). Although QUIC is mainly designed to be optimized for the newWeb communication (HTTP/3), QUIC created by the IETF is also intended for applications other than the Web. To investigate the performance of QUIC as a transport layer, we evaluate the performance of a method implemented on QUIC based on FTP, a file transfer protocol, and discuss the characteristics of QUIC.

Publication History:

Received: March 29, 2023
Accepted: April 09, 2023
Published: April 10, 2023

Keywords:

QUIC, File Transfer, Performance

Introduction

The amount of data exchanged on the Internet today is enormous, and attempts have been made to reduce the amount of traffic flowing over the network and to minimize communication latency. In November 2009, Google proposed a new communication protocol called SPDY[1], which attempts to speed up communication by adding features such as multiple connections and pipelining to HTTP/1.1. Since then, SPDY has gone through several iterations and is now standardized by RFC 9113[1] under the name HTTP/2. However, because SPDY and its successor HTTP/2 were Transmission Control Protocol (TCP)-based protocols, they could not overcome the latency that occurs at the TCP layer.

To solve the above issues in SPDY and HTTP/2, a transport protocol called QUIC was proposed by Google in 2013. QUIC is designed on top of UDP with multiple connections, connection management functions equivalent to TCP, and packet management functions. This overcame the latency caused by TCP, which could not be solved by SPDY or HTTP/2. QUIC has had several iterations and was standardized by the Internet Engineering Task Force (IETF) in May 2021 as RFC 9000 [2]. HTTP/3, which relies on QUIC, was also standardized as RFC 9114 [3] in June 2022.

On the other hand, there have been active attempts to make existing application protocols available over QUIC, and in December 2020, Internet-Draft[2], an improved version of SSH that can be used on QUIC, was proposed. In addition, Microsoft has implemented SMB over QUIC on Windows Server 2022[3], and the groundwork is being laid for further QUIC adoption.

In this study, we propose an application of QUIC to file transfers, and aim to realize secure file transfers with lower latency than existing methods. We also compare the performance of QUIC with existing file transfer protocols and evaluate its performance.

QUIC

QUIC (Quick UDP Internet Connections) is a User Datagram Protocol (UDP)[4] based transport protocol standardized by RFC 9000 [2]. Unlike TCP, UDP does not have features such as retransmission control, sequence control, and congestion control, all of which are implemented in QUIC. This gives QUIC the following advantages.

- Avoiding HoL Blocking at the Transport Layer Level
- Transport layer level encryption
- Realization of multiplexed communication by streaming

When a packet loss occurs in TCP, the operating system first asks the endpoint to retransmit the packet. Then, after the packet loss is recovered, the data is passed to the application. At this point, even if subsequent packets are received, the data cannot be processed until the packet loss is recovered. This is called HoL Blocking. On the other hand, QUIC can efficiently process data even in the case of packet loss, because it can pass subsequent packets to the application. If application layer data needs to be encrypted, encryption methods such as TLS over TCP can be used. Since TLS cannot encrypt information related to the control of the communication path, a malicious third party can eavesdrop on the communication state. In QUIC, however, TLS is absorbed by QUIC itself and used internally. Therefore, information related to the control of the communication paths can also be encrypted. QUIC can have multiple communication management units called ”streams” on a single connection. Each stream is reliable on a per-stream basis, and packet loss recovery and sequencing can be performed without affecting other streams. This allows data to be processed more efficiently than with TCP. Each stream is assigned an integer value called a stream ID to uniquely identify the stream. Stream IDs are numbered according to the following rules (Table 1).

Start side	Stream direction	Lower 2 bits	Example of Stream ID
Client	Bi-directional	00	0, 4, 8, 12, · · ·
Server	Bi-directional	01	1, 5, 9, 13, · · ·
Client	Uni-directional	10	2, 6, 10, 14, · · ·
Server	Uni-directional	11	3, 7, 11, 15, · · ·

Table 1: Stream ID numbering rules.

Related Work

Nepomuceno et al. [5] conducted experiments to measure the time to retrieve a Web page using HTTP for each of QUIC and TCP, and

***Corresponding Author:** Prof. Tomofumi Matsuzawa, Department of Information Sciences, Tokyo University of Science, Tokyo 162-8601, Japan, E-mail: t-matsu@is.noda.tus.ac.jp

Citation: Ishikawa Y, Matsuzawa T (2023) Evaluation of a File Transfer Protocol with QUIC. Int J Comput Softw Eng 8: 184. doi: https://doi.org/10.15344/2456-4451/2023/184

gave a quantitative evaluation. The experiment first uses Alexa's page access ranking to determine the top 100 pages to be used for the test. Then, the tool Mahimahi is used to record HTTP (HTTPS) traffic. Under the network traffic condition by Wang et al. research [6], we replay the page fetches on the network and measure the time it takes to load the page. Experiments are also conducted with and without web browser caching enabled. The experimental results show that RTT variations have a significant impact on QUIC performance, while TCP performance is not significantly affected, and packet loss rate has no significant impact on either performance. The paper also shows that QUIC outperformed TCP in each test by less than 40%. Furthermore, with caching enabled, TCP performed better than QUIC. The paper attributes this to the rendering engine's inability to handle QUIC efficiently and the fact that the web pages used in the experiments were not optimized for QUIC.

Proposed Method

Memory constraints

In this study, we propose to apply QUIC to secure multiple file transfers, taking advantage of QUIC's capability for multiplexed connections and encrypted communications. We implement a prototype of a file transfer protocol that runs on QUIC, based on the FTP[7] specification.

The following procedure is used to initiate communication.

1. The QUIC file transfer client (hereinafter referred to as "client") establishes a QUIC connection to the server. The port used is UDP N (N is known to the client).
2. A bidirectional stream is opened from the client to the QUIC file transfer server (hereinafter referred to as "server"). This stream is called the "control stream". The control stream is used to control data communication.

The following procedure is used to initiate communication.

1. The client sends a comma-separated list of the names of the files it requests to be transferred as an ASCII string to the server.
2. The server concatenates the names of the requested files that can be transferred with commas and sends them to the client. If the file does not exist, an empty string is sent.

A schematic diagram of a file request is shown in Figure 1.

In the list of received file names, files that do not exist on the server or that are inaccessible due to permission issues are omitted from the list, and only files that can be transferred are reported to the client.

The following procedure is used for file transfer, similar to the active mode in FTP.

1. The server opens a bi-directional stream for the number of files to be transferred. This stream is called a data stream. Data is transferred using the data stream from this point on.
2. The client creates threads for the number of files to be received and performs the following processing.

(a) Opens the destination file.
(b) Reads a sequence of bytes from the stream.
(c) Writes the read bytes to the file.
(d) Closes the file.
(e) Closes the data stream from the client side.

Experiments and Discussion

Both the client and server programs are implemented in the Go programming language. The library for handling QUIC packets is quic-go[4], created by L. Clemente.

Table 2 shows the specifications of the computer used as the experimental environment.

	Client	Server
OS	Ubuntu 22.04.01 LTS	Debian 11
CPU	Intel Core i7-8550U	Broadcom BCM2711
RAM	20GB	8GB

Table 2: Specifications of the computer.

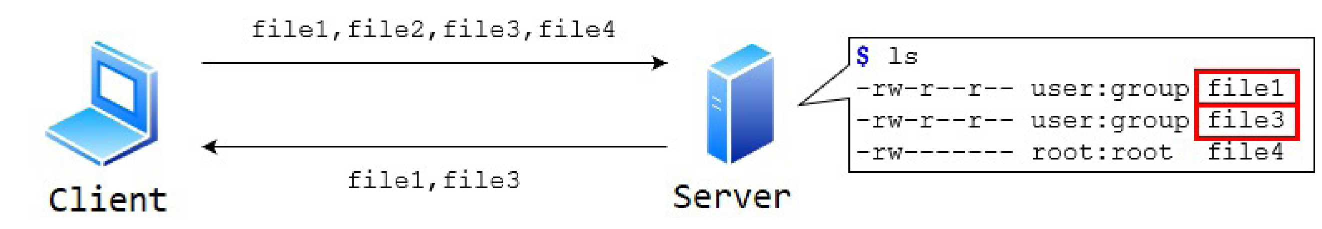
All experiments were conducted within the local network, and fault elements such as packet loss and packet delivery delays were reproduced by intentionally generating them using the tc command. In our experiments, we measure the throughput of transferring small files in parallel and the throughput of transferring a single large file. For this purpose, the following two types of files are prepared.

- **File group 1**: 100 JPEG files

  -Minimum: 177KB
  -Maximum: 1.72MB
  -Average: 658KB
  -Standard Deviation (SD): 407KB

- **File group 2:** 1 ISO file

  – Size: 1.47GB



```
$ ls
-rw-r--r-- user:group file1
-rw-r--r-- user:group file3
-rw------- root:root  file4
```

Figure 1: Schematic of file request.

We perform packet capture using Wireshark and measure the time required to transfer the above file groups 1 and 2 with the implementation of the proposed method according to the conditions in Table 3. [5] The same conditions are also used in the SCP[6] experiment for comparison.

	Fault Elements File group	File group
QUIC	-	1
QUIC	Delay 5ms	2
QUIC	Delay 10ms	1
QUIC	Delay 15ms	2
QUIC	Delay 20ms	1
QUIC	Delay 25ms	2
QUIC	Loss ratio 1%	1
QUIC	Loss ratio 2%	2
QUIC	Loss ratio 3%	1
QUIC	Loss ratio 4%	2
QUIC	Loss ratio 5%	1

	Fault Elements File group	File group
SCP	-	1
SCP	Delay 5ms	2
SCP	Delay 10ms	1
SCP	Delay 15ms	2
SCP	Delay 20ms	1
SCP	Delay 25ms	2
SCP	Loss ratio 1%	1
SCP	Loss ratio 2%	2
SCP	Loss ratio 3%	1
SCP	Loss ratio 4%	2
SCP	Loss ratio 5%	1

Table 3: Experimental conditions.

Figure 2 shows the time required to transfer file group 1 when the fault elements are set according to Table 3.

On the horizontal axis, packet loss rate is expressed in %, and packet delay is expressed in ms.

When a fault factor is set, the implementation of the proposed method does not show a significant effect on the transfer time. On the other hand, the transfer time increases with the size of the fault in SCP.

The transfer time of the proposed method using QUIC increases linearly with increasing packet loss rate in the order of about 0.1 second. On the other hand, in the comparison experiment with SCP, the transfer time increases with increasing packet loss rate by an order of magnitude larger than the first power of the packet loss rate. In addition, the transfer time of the proposed method is almost constant even when the delay increases, while the transfer time of SCP increases linearly with a unit of about 1 second. Based on the above results, it can be said that the proposed method improves the throughput of file group 1.

Experimental results show that the proposed method performs better than the existing methods in environments where packet loss and packet delivery delays occur. In particular, the difference in transfer time between the proposed and existing methods becomes larger as the network disturbance factor increases. This is due to the fact that the overhead caused by HoL blocking exceeds the overhead caused by asynchronous processing as the packet loss rate increases and the number of seconds of delay in packet delivery increases. Therefore, the proposed method is superior for the case where many files are transferred simultaneously in an environment with faulty elements.

When there is no faulty element, the file transfer time using the proposed method is slightly longer. This is due to the processing time of the computer. The proposed method processes multiple streams asynchronously when downloading multiple files, which increases the load on the computer compared to existing methods. In our experiments, this asynchronous processing affected the results. However, since it is rare for real networks to be free of such obstacles, this performance difference can be ignored as a practical matter.

In light of the above, the proposed method is expected to perform well when transferring a large number of files in a real network environment.

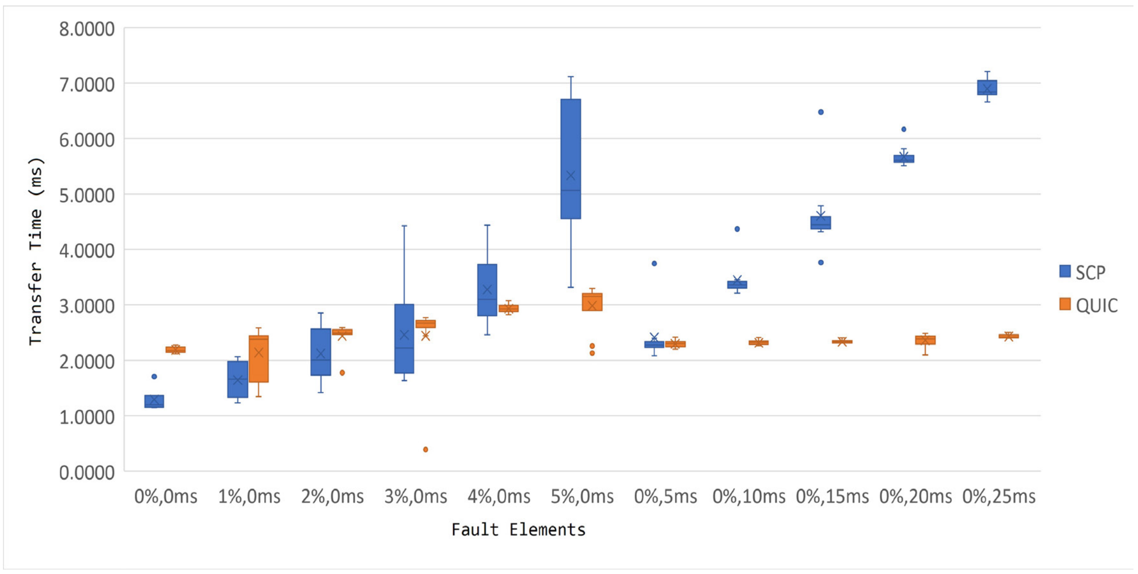


Figure 2: Fault element and transfer time for file group 1.

[5]The time from the start of the handshake to the disconnection of communication is defined as the time required for the transfer.

[6]An application protocol for file transfers using SSH (Secure Shell), an encrypted remote login system.

Figure 3 shows the time required for the transfer of file group 2 when the fault elements are set according to Table 3.

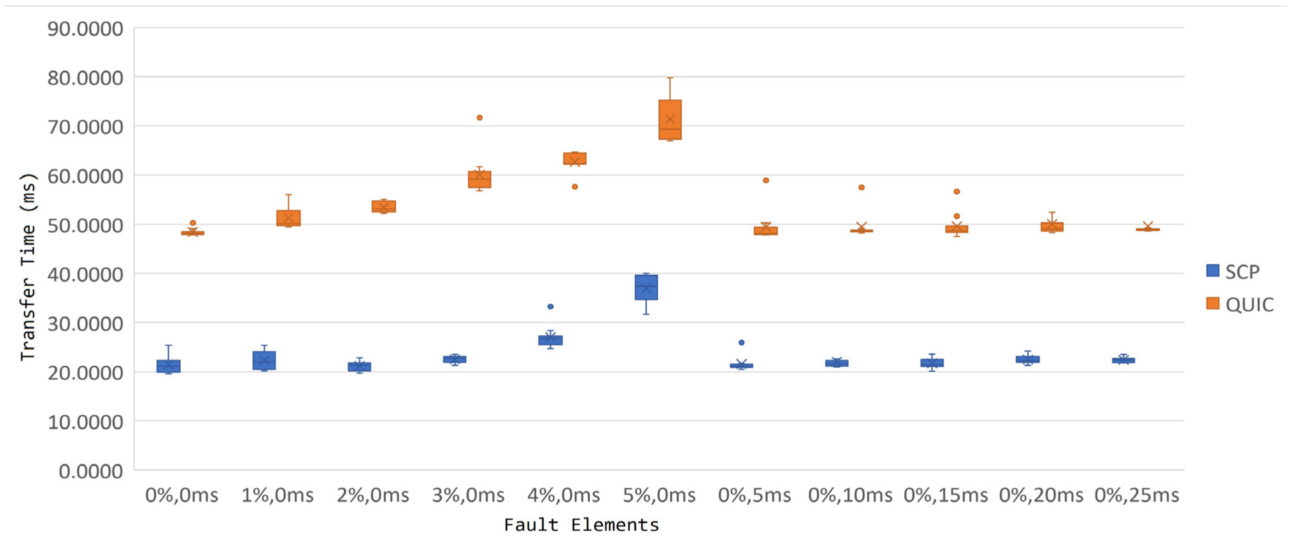


Figure 3: Fault element and transfer time for file group 2.

In both cases, the transfer time of the proposed method is about 30 seconds longer than that of SCP. For both the proposed method and SCP, the transmission time increases as the packet loss rate increases. On the other hand, the transfer time is almost constant for both the proposed method and SCP. In all conditions, SCP was 30 seconds faster for file group 2, and the proposed method did not improve the throughput. From the experimental results, it can be seen that the proposed method requires 30 seconds more transfer time regardless of the presence or absence of error elements. In this experiment, New Reno was used as the congestion control algorithm, and the size of the send/receive buffer was set to 21,299,200 bytes (≃ 20.3 MiB). So, it is clear that these factors are irrelevant to the results.

According to the packet capture results, 924,454 SSH packets were sent and received. On the other hand, 1,340,576 QUIC packets were sent and received, which is about 1.5 times the number of SSH packets. In this experiment, it is assumed that the time required to encrypt and decrypt a large number of QUIC packets has greatly affected the transfer time.

Based on the above, the proposed method is not suitable for transferring large files, at least in our implementation.

Conclusions

In this study, we proposed a fast and highly secure file transfer protocol that exploits the low latency and multiple connectivity of QUIC for file transfers. In this study, we implemented a prototype file transfer protocol running on QUIC using existing libraries.

Experiments were conducted to measure the throughput of small files transferred in parallel and the throughput of large files transferred in one shot. Similar experiments were also performed on SCP, an existing method, to compare the throughput of the two methods. The experimental results showed the superiority of the proposed method over the existing method in the former case, but not in the latter case.

While many studies have discussed the superiority of QUIC in the context of HTTP, this study evaluated the performance of QUIC outside the context of HTTP. This study is significant in the recent trend of various application protocols supporting QUIC.

Competing Interests

The author declare that he has no competing interests.

References

1. Thomson M, Benfield C (2022) "HTTP/2", RFC 9113.
2. Iyengar J, Thomson M (2021) QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000.
3. Bishop M (2022) "HTTP/3", RFC 9114.
4. Postel J (1980) User Datagram Protocol, RFC 768.
5. Nepomuceno K, de Oliveira IN, Aschoff RR, Bezerra D, Ito MS, et al. (2018) "QUIC and TCP: A Performance Evaluation". IEEE Symposium on Computers and Communications (ISCC), pp. 00045-00051.
6. Wang XS, Balasubramanian A, Krishnamurthy A, Wetherall D (2014) How speedy is spdy?", The 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI).
7. Postel J, Reynolds J (1985) File Transfer Protocol (FTP), RFC 959.