Firewall Traversal Method by Inserting Pseudo TCP Header into QUIC

Keigo Taga, Junjun Zheng, Koichi Mouri, Shoichi Saito, Eiji Takimoto

***Abstract*—A wide range of communication protocols has been developed recently to address service diversification. At the same time, firewalls(FWs) are installed at the boundary between internal networks such as those owned by companies and homes, and the Internet. In general, FWs are configured as whitelists is whitelist that release only the port corresponding to the service to be used and block communication from other ports. This means that many protocols except those well used are blocked by FWs resulting in users not being able to benefit from any new protocols. In this paper, we propose a method for traversing an FW and enabling communication by inserting a pseudo TCP header imitating HTTPS into a packet, which will be blocked by the FW. Since the packet capsulated by the proposed method disguised by HTTPS camouflaging only when passing through the FW, the TCP control of the end node is not executed, and the advantages of Quick UDP Internet Connection are not lost. In this study, we implemented the proposed method as a loadable kernel module using Netflter in Linux and verified its operation and performance.**

***Index Terms*—QUIC, Firewall, TCP, Capsuling.**

I. Introduction

A wide range of communication communication protocols has been developed recently to address service diversification. Improvement of service quality and communication performance can be expected when using these communication protocols. At the same time, firewalls(FWs) are installed at boundary between internal networks such as those owned by companies and homes, and the Internet. An FW, which is a system for ensuring security, guarantees security by limiting communication protocols and ports. Therefore, even if the end nodes support a specific protocol, communication may be blocked by an FW. As a result, many clients may not be able to use services developed by service providers based on newly developed protocols due to an FW. In other words, an FW may become a barrier preventing the spread of new protocols and improvement of service quality .

In such a case, the FW configuration may be changed to open the port used for communication. However, changing the FW configuration is not preferable from the viewpoint of network management. Thus, many organizations prohibit Quick UDP Internet Connections(QUIC) using UDP as the transport layer protocol and RTP communication, whose port number to be used is determined dynamically.

Tunneling is often used as a method for enabling the use of communication protocols that are not permitted by an FW without changing the FW configuration.

This study was supported by the Grants-in-Aid for Scientific Research of the Japan Society for Promotion of Science (JSPS) under the Contract No. JP18K18045.

K. Taga, J. Zheng, K. Mouri, and E. Takimoto are with Ritsumeikan University (e-mail: ktaga@asl.cs.ritsumei.ac.jp).
S. Saito is with Nagoya Institute of Technology.

Tunneling encapsulates original packets by other protocol. Ordinal tunneling uses HTTP or HTTPS for capsulation because these protocols are the most widely used. They use TCP as the transport layer protocol. Even if tunneled communication uses another transport layer protocol, TCP influences the communication much stronger than the other protocol. For example, assume that QUIC is the target of HTTP tunneling. QUIC uses UDP and has its own TCP-like control mechanisms. When a packet loss occurs, TCP of HTTP tunneling retransmits transparently from QUIC. In this way, TCP mechanisms work prior to the ones of QUIC. That is to say, QUIC over HTTP cannot exhibit its performance.

In this study, we propose an FW traversal method, which inserts a pseudo TCP header with the purpose of realizing communication without affecting it on a communication path, where delivery of packets between end nodes is not guaranteed. The proposed method makes it possible to use various communication protocols without being restricted by an FW. The method achieves this by disguising packets that use protocols and port numbers usually restricted by an FW making them look as if they are parts of HTTPS communication. Impersonation of HTTPS communication is achieved by encapsulating the pseudo TCP specified port 443 for the payload of the IP datagram of the target packet. By labeling a packet to be a part of HTTPS traffic, it becomes possible to exclude it from the FW targets to be filtered. In addition, the proposed method does not affect the control of the protocol that the FW wishes to pass since the method only rewrites the packet on the communication path preventing the TCP control from working. The proposed method inserts the pseudo TCP header, discards it after passing through the FW, and returns it to the original packet to obtain communication.

In this paper, we discuss application of the proposed method to QUIC developed by Google and expected to be popular in the future. QUIC is a transport layer protocol that operates on the user land; it is designed and developed on the premise of combination with HTTP/2[1].

QUIC communicates using UDP/443. Therefore, the proposed method inserts and discards the pseudo TCP header for the communication of UDP/443 port. It sets the port number of the pseudo TCP header to be inserted to 443, making it look like HTTPS communication. Furthermore, similar to TCP communication, emulation processing of three-way handshake is performed when communication start of QUIC and connection migration occurs. QUIC is a protocol that performs acknowledgment similar to TCP; hence, pseudo TCP communication ca be seen as acknowledged after inserting a pseudo TCP header with the ACK flag set for the QUIC packet.

In this study, we implemented the proposed method as a loadable Kernel Module(LKM) using Netflter in Linux, verified and evaluated its performance for QUIC. The module verification results confirm that communication is possible via a stateful inspection FW using the proposed method.

The rest of this paper is organized as follows. An overview of related research is provided in Section 2. The proposed FW traversal method that inserts a pseudo TCP header is presented in Section 3. Application of the proposed method to QUIC and its evaluation are described Section 4 and 5, respectively. Conclusion and future work are provided in Section 6.

II. Related work

A. *Retransmission-Controlled TCP*

Retransmission-Controlled TCP[2] suppresses retransmission control of TCP to perform real-time communication. While the behavior of TCP controlling retransmission is close to that of UDP, controls other than the retransmission control work; hence, performance of the protocol to be passed is affected in Retransmission-Controlled TCP.

B. *SoftEther*

SoftEther[3] is a layer 2 VPN protocol encapsulating an Ethernet frame and transmitting a packet. Since HTTPS is used as a tunneling protocol, SoftEther makes it possible to pass through an FW, which is encrypted after the TCP header and checks up to the application layer in detail. However, SoftEther uses TCP connection for transmission. Therefore, similar to the Retransmission-Controlled TCP, each control mechanism of TCP exerts an impact on the control of the protocol to be passed.

C. *SOCKS*

SOCKS[4] is a technology that relays communication of the transition layer protocol and passes through an FW. In SOCKS, a proxy server is installed at the boundary between the external and networks, and external and internal communication is accepted. Upon receiving the connection, the SOCKS server authenticates the end user as necessary, notifies, and connects to the destination node. Application correspondence is necessary for using SOCKS; it cannot be used transparently. In addition, there is problem of throughput degradation since the SOCKS server receives and retransmits, a packet once.

III. Proposed method

In this study, we propose an FW traversal method that inserts a pseudo TCP header to obtain communication of a specific protocol on a communication path where packet delivery between end nodes is not guaranteed. The proposed method makes it possible to use various communication protocols without being restricted by an FW. The method achieves this by disguising packets that use protocols and ports usually blocked by an FW making them look as if they are part of HTTPS communication. Impersonation of HTTPS communication is achieved by encapsulating the pseudo TCP specified port 443 for the payload of the IP datagram of the target packet. In this case, the pseudo TCP performs only three-way handshake and connection management as the control of TCP. In the proposed method, a pseudo TCP header is inserted into a packet for encapsulation and transmitted. The transmission packet can be returned to the original form by removing the pseudo TCP header after the packet passes through an FW. Unlike the existing technology, double control does not occur since various control mechanisms of TCP do not work in pseudo TCP. That is, FW traversal is realized using encapsulation without impairing the superiority of the target protocol.

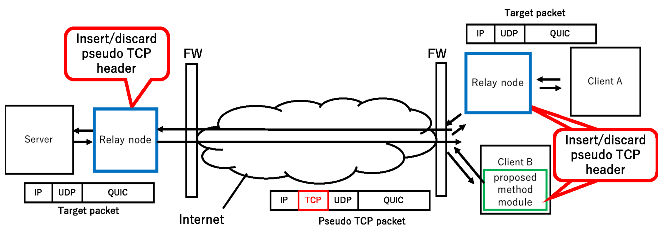


Fig. 1. System image of proposed method.

Fig. 1 depicts a system image of the proposed method. It is necessary to install the mechanism that implements the proposed method before and after the FW that blocks communication. Several patterns can be considered for the installation location of the proposed method. In this study, we assume that the proposed method is installed in each network to which the client and the server belong, as shown in Fig. 1. The proposed method mechanism can be installed on the end node such as the client or server, a terminal existing on the communication route such as a router of each network, or the same network such as the proxy. As a prototype, we assume here that a relay node having the proposed mechanism is installed for each network to which the client and server belong.

For the communication applying the proposed method to pass through a stateful inspection FW, it is necessary to set each field value of the pseudo TCP header to be inserted by emulating TCP communication. Therefore, the proposed method emulates three-way handshake at the start of communication and periodically acknowledges during communication. Fig. 2 depicts the emulation of three-way handshake in the proposed method. When the proposed mechanism on the client side receives the start packet of the new flow of the target protocol, it the proposed method mechanism makes the packet to wait. Next, the proposed mechanism sends and receives SYN, SYN/ACK, and ACK packets. A flags is set in the reserved field of the TCP header distinguish these packets from the actual TCP three-way handshake packet. In addition, these communication types three-way handshake between client servers using the IP addresses of the clients and servers. Our mechanism then performs a pseudo TCP header insertion process on the packet that was kept waiting and transmits it.

The proposed, mechanism establishes a connection with the FW existing on the communication path, makes it appear as HTTPS communication, and allows packets to pass through the FW. Moreover, the proposed method can disguise communication as something other than HTTPS communication by changing the port number of the pseudo TCP header to be inserted to another one. In addition, application of the proposed method can aggregate and convert the ports used by the target protocol; therefore, it is effective for protocols that randomly select ports from their a wide range.

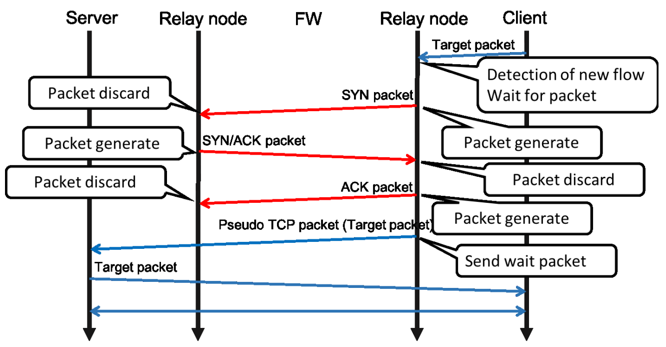


Fig. 2. Three-way handshake emulation of the proposed method.

IV. Application of the Proposed Method to QUIC

QUIC is a transport layer protocol that operates on the user land and can handle multiple streams simultaneously. This protocol is designed and implemented on the premise of a combination with HTTP/2. The protocol communicates using UDP rather than TCP or TLS that were used in conventional Web communication. Therefore, QUIC performs TCP control, TLS encryption, authentication processing, HTTP/2 stream management, and flow control. The protocol unifies these processes to make it specialized in the HTTP/2 transport layer protocol, thereby trying to reduce the communication delay associated with them. Currently, QUIC is being standardized by the Internet Engineering Task Force.

Since QUIC allows to reduce communication delay, page loading time can be reduced. As a result, Web service providers can develop a wider range of services while improving the user experience of the client with enhanced communication performance using QUIC. However, UDP/443 used by QUIC is blocked in many FW configrations. Therefore, even if QUIC is enabled for both the client and server, TCP and TLS are usually used instead of QUIC. To solve this problem, we implement the proposed method for QUIC communication.

To build the proposed method, it is necessary to modify the packet on the communication route. Therefore, we implemented the proposed method in Linux as an LKM using Netfilter. Netlfiter is a framework for obtaining packet filtering and network address translation function. In Netfilter, hook points are defined for each protocol in the packet processing routine, and it is possible to rewrite hooked packets by registering a callback function at the hook point. In this section, we describe the processing of the proposed method for QUIC.

A. Client-side LKM processing

The client-side LKM hooks packets at the IP layer and processes the proposed method. First, it judges whether the hooked packet is a packet related to the proposed method, which could be a QUIC, pseudo TCP, or SYN/ACK packet with a flag in the reserved field. When the hooked packet is a pseudo TCP packet, the LKM discards a pseudo TCP header, updates the IP header, and returns the packet to the hook point. The packet is judged as a pseudo TCP packet if TCP and the source port number is 443, and the value of the first four bytes of the TCP header matches the value of the first four bytes of the UDP header. When the hooked packet is a QUIC packet, a pseudo TCP header is inserted between the IP and UDP headers. If it is a UDP packet of the destination port 443, then it is a QUIC packet. Next, the LKM judges whether or not the hooked QUIC packet is a new flow. Four tuples of IP address and port number are used for flow judgement. When it is a known flow, the QUIC packet, in which the pseudo TCP header is inserted, is returned to the hook point and the packet is transmitted. When the flow is a new flow, the pseudo TCP packet is not returned to the hook point but kept on standby; further the emulation process of the three-way handshake is then started. When the hooked packet is a SYN/ACK packet with a flag in the reserved field, it transmits an ACK packet with a flag in the reserved field to the server and discards the hooked SYN/ACK packet. When the hooked packet is not one of the above-mentioned three packet types, it is not modified and returned to the hook point.

B. Server-side LKM processing

The server-side LKM hooks IP packets with NF_IP_PRE_ROUTING and processes the proposed method. First, it judges whether the hooked packet is a packet related to the proposed method. In the server-side LKM, the packet related to the proposed method can be a QUIC packet, pseudo TCP packet, SYN or ACK packet with a flag in the reserved field. When the hooked packet is a pseudo TCP packet, the same processing is performed as by the client-side LKM. The packet is determined as begin a pseudo TCP if TCP and the destination port number is 443, and the value of the first four bytes of the TCP header matches the value of four bytes immediately after the TCP header. When the hooked packet is a QUIC packet, the same processing is performed as by the client-side LKM. However, the emulation processing of three-way handshake is not performed even for undetected flows. When the hooked packet is a SYN packet with a flag in the reserved field, it transmits a SYN/ACK packet flagged in the reserved field to the client and discards the hooked SYN packet. When the hooked packet is an ACK packet with a flag in the reserved field, the hooked ACK packet is discarded. When the hooked packet is not one of the above-mentioned four packet types, it is not rewritten and returned to the hook point.

C. Emulation of sequence and acknowledgment numbers

To enable passing through a stateful inspection FW, each field value of the pseudo TCP header to be inserted needs to be set by emulating actual TCP communication. Therefore, we implemented a process emulating the sequence and acknowledgment numbers of the pseudo TCP header.

The initial sequence number of the pseudo TCP communication of the proposed method is generated using a random number when a SYN or SYN/ACK packet is generated at the time of emulating three-way handshake. The sequence number of the ACK packet stores the value of the sequence number + 1 stored in the SYN packet. The acknowledgment number of the SYN/ACK and ACK packets stores the value of the sequence number + 1 stored in the packet that triggers these packet generation and transmission processing.

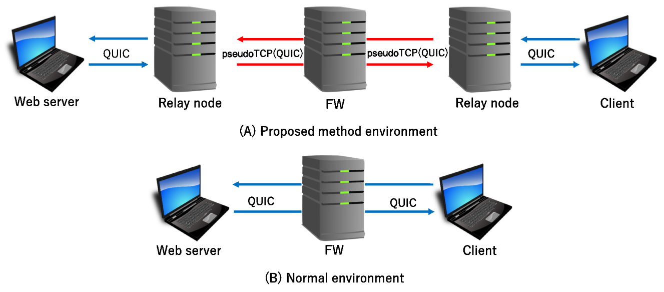
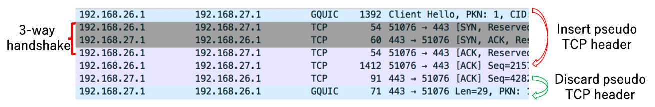


Fig. 3. Evaluation environment.

```
*filter
:INPUT ACCEPT
:FORWARD DROP
:OUTPUT ACCEPT
-A FOWARD -p tcp ! --syn -m conntrack --ctstate NEW
 -j DROP
-A FOWARD -m conntrack --ctstate INVALID -j DROP
-A FOWARD -p tcp --sport 80 -j ACCEPT
-A FOWARD -p tcp --dport 80 -j ACCEPT
-A FOWARD -p tcp --sport 443 -j ACCEPT
-A FOWARD -p tcp --dport 443 -j ACCEPT
-A FOWARD -p icmp -j ACCEPT
COMMIT
```

Fig. 4. Configuration of iptables.



```
192.168.26.1    192.168.27.1    GQUIC   1392 Client Hello, PKN: 1, CID
192.168.26.1    192.168.27.1    TCP       54 51076 → 443 [SYN, Reserve
192.168.27.1    192.168.26.1    TCP       60 443 → 51076 [SYN, ACK, Re
192.168.26.1    192.168.27.1    TCP       54 51076 → 443 [ACK, Reserve
192.168.26.1    192.168.27.1    TCP     1412 51076 → 443 [ACK] Seq=215
192.168.27.1    192.168.26.1    TCP       91 443 → 51076 [ACK] Seq=428
192.168.27.1    192.168.26.1    GQUIC     71 443 → 51076 Len=29, PKN:
```

Fig. 5. Packet capture log of client-side relay node.

After completing the emulation process of three-way handshake, the sequence and acknowledgment numbers of the packet with inserted pseudo TCP header waiting on the client-side LKM store the same value as the last ACK packet upon three-way handshake. The sequence number after completing three-way handshake stores the sequence number + the value of the TCP payload length stored in the pseudo header of the packet transmitted in the same flow. With this processing, the sequence number of the pseudo TCP communication of the proposed method increases sequentially for each TCP similar to the actual TCP communication.

The acknowledgment number of the pseudo TCP header is determined by the sequence number of the pseudo TCP header that was discarded last. In particular, the acknowledgment number of the pseudo TCP header stores the sequence number + the TCP payload length of the packet with inserted pseudo TCP header that was discarded last in the same flow. In addition, the ACK flag is set in all pseudo TCP headers during data communication.

V. Evaluation

In this section, we describe functional and performance evaluation of the proposed FW traversal method that inserts pseudo TCP header. In function evaluation, we confirm that QUIC communication is possible when applying the proposed method under an FW environment setting to shut down QUIC communication. In the performance evaluation, we confirm the overhead caused by applying the proposed method and its impact on the page loading time.

A. *Evaluation environment*

Fig. 3 shows the evaluation environment. The difference between Figs. 3(A) and 3(B) is in the proposed method. In the client, QUIC's test client software proto-quic[5] works. In the Web server, Caddy[6] operates. Caddy is web server software compatible with Google version of QUIC. The relay node performs the processing of the proposed method. We built an FW that blocks QUIC communication using iptables, which is the Linux standard FW. Fig. 4 shows the configuration of iptables, where all packets other than TCP/80, 443, and ICMP packets are blocked. When an error is detected in the header of the packet, the packet is discarded, even if it is a TCP/80 or 443 packet using state matching of the conntrack module.

B. *Functional evaluation*

In this evaluation, we confirm that QUIC communication can be obtained by passing through FWs that block QUIC communication. Fig. 5 represents a packet capture log of the client-side relay node, which shows that a communication start packet is made to wait, emulation of three-way handshake is performed, a pseudo header is inserted for the QUIC packet transmitted from the client, and the pseudo TCP header of the received pseudo TCP packet is discarded. Fig. 6 represents the output log of the proto-quic client. Since the HTTP status record of 200 has been returned from the Web server, the communication is successful. Therefore, we conclude that QUIC communication becomes possible when using using the proposed method under the FW environment with the configuration shown Fig. 4.

C. *Overhead measurement*

Since the proposed method performs processing such as insertion and discarding of pseudo TCP headers, an overhead occurs. To measure this overhead, the same communication was performed as represented in Figs. 3(A) and 3(B).

As a measurement method, we performed QUIC communication in both environments and packet capture on the client. The RTT is measured from the timestamp of a packet. The start time is when a QUIC packet is transmitted from the client, whereas the end time is when the client receives the response packet. In the environment shown in Fig. 3(B), which excludes the relay nodes for applying the proposed method, measurements are carried out in the same way, and the increment of RTT is measured by applying the proposed method. handshake, the QUIC packet of the new flow with standby processing is excluded from measurement.

The RTT increased by 0.92ms when the proposed method was applied. Therefore, the delay per packet is 0.46ms due to adding two relay nodes and the pseudo header insertion and discarding processing of the proposed method.

D. *Measurement of page loading time*

In the section V-C, we confirmed that delay of 0.46ms is observed per packet when applying the proposed method.

```
Response:
headers:
{
  :status 200
  alt-svc quic=":443"; ma=2592000; v="38,37,36,35"
  server Caddy
  etag "pcvcl73y0s"
  last-modified Fri, 03 Aug 2018 05:00:43 GMT
  content-type text/html; charset=utf-8
  accept-ranges bytes
  content-length 184060
}
```

Fig. 6. Output log of proto-quic client.

TABLE I
MEASUREMENT RESULT OF RTT(MS).

Enviroment	Fig. 3(A)	Fig. 3(B)
RTT	2. 36ms	1. 42ms

Therefore, we measure the effect of the increased delay on communication and page load time.

1) Measurement method: To investigate the impact on the Web page loading time, we used QUIC to communicate in the environment of Fig.3(A) where a relay node was introduced to apply the proposed method, and environment of Fig.3 (B), where no relay terminal exists. We prepared three html files of 1, 5, and 10MB to measure the loading times. To simulate the actual Internet, the FW terminal generated a communication delay by using the tc command. There are five types of delays: no delay, 20, 50, 70, and 100ms. We measured the delay 10 times and calculated its average value. In the proposed method, the three-way handshake emulation processing is performed at the beginning of communication. Therefore, we separately measured the time taken to emulate three-way handshake. The time required for reading a page was measured from the timestamp of the packet captured at the client. The measurement started at the first captured QUIC packet and ended up to the QUIC packet last received by the client during the communication. The time required for the three-way handshake of the proposed method was measured from the timestamp of the packet captured at the client-side relay node. The measurement started when the QUIC packet was received first and end when the emulation process of three-way handshake ended and the packet with inserted pseudo TCP header was transmitted.

2) Result: Table II shows the measurement results for the page loading times in the normal environment of Fig.3(B) and in the environment, using the proposed method represented in Fig.3(A). It can be noticed from the table that the time required for the emulation processing of three-way handshake by the proposed method is 3WH. This result is explained in detail in Section 5-E.

E. Discussion

We confirmed that the proposed method allows QUIC communication to pass through a stateful inspection FW. In addition, we confirmed that the proposed method increases the one-way latency per packet by 0.46ms. However, the impact of the proposed method on the page loading time was small for QUIC transmitting and receiving packets continuously. It is clear that the value obtained by subtracting 3WH from the results achieved by the proposed method is close to that achieved for the normal environment (see Table 2). On the other hand, the emulation processing of three-way handshake of the proposed method had a big influence on the page loading time. The time required for 3-way handshake of the proposed method depends on RTT between relay nodes. Therefore, it is thought that the impact of the proposed method would be noticeable in networks with large RTT. However, three-way handshake is performed only once when QUIC communication is started. Therefore, it is expected to have limited effect since the increase in the communication time caused by three-way handshake is inversely proportional to the total communication time of QUIC. This is clear from the results shown in Table II. In addition, the impact of three-way handshake is consistent with RTT between relay nodes and it is constant depending on the state of the network. The impact of three-way handshake is thought to decrease as communication time increases. Actually, the time required for three-way handshake was almost equal to the set communication delay. Furthermore, as a result of the communication delay of 100ms, the ratio of communication time in the environment of Figs. 3(A) and Fig. 3(B) is 1.125 when the file size is 1MB and 1.0476 when it is 10MB.

TABLE II
MEASUREMENT RESULT OF PAGE LOAD TIME(MS).

		0ms	20ms	50ms	70ms	100ms
1MB	Fig. 3(B)	79.0	189.0	419.7	581.5	824.3
	Fig. 3(A)	77.9	213.5	477.2	656.5	927.4
	3WH	0.63	20.7	50.78	70.83	100.7
5MB	Fig. 3(B)	379	565	835	962	1304
	Fig. 3(A)	398	617	893	1064	1429
	3WH	0.63	20.7	50.8	70.7	100.6
10MB	Fig. 3(B)	725	843	1183	1330	1798
	Fig. 3(A)	701	841	1244	1426	1881
	3WH	0.66	20.7	50.8	70.7	100.8

VI. CONCLUTION

In this paper, we proposed an FW traversal method that inserts a pseudo TCP header with the purpose of realizing communication on a communication path where packet reachability between end nodes is not guaranteed. The proposed method enables various communication protocols to be used without being restricted by an FW. Moreover, the proposed method dose not affect the control of the target protocol since it only rewrites the packet on the communication path.

We implemented and evaluated the proposed method on Linux using Netfilter for QUIC that is expected to become popular in the future. We confirmed that the target communication can pass through a stateful inspection FW when applying the proposed method. Moreover, we confirmed that only the emulation process of three-way handshake implemented in our method impacts the page download time. In the future, we plan to apply the proposed method to communication on the Internet and verify its effectiveness.

REFERENCES

[1] M. Belshe, R. Peon, and M. Thomson, “Hypertext Transfer Protocol Version 2 (HTTP/2),” RFC 7540, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt

[2] S. YOKOYAMA, H. YAMAMOTO, and K. YAMAZAKI, “The evaluation of communication characteristic of cellular network and implementation and evaluation of retransmission-controlled tcp,” *The IEICE transactions on information and systems (Japanese edition)*, vol. 95, no. 5, pp. 1133–1141, may 2012.

[3] D. Nobori, “Virtual ethernet system and tunneling communication with softether,” https://www2.softether.jp/jp/company/media/academic/data/softetherpaper001.pdf, jan 2004.

[4] M. D. Leech, “SOCKS Protocol Version 5,” RFC 1928, Mar. 1996. [Online]. Available: https://rfc-editor.org/rfc/rfc1928.txt

[5] “proto-quic,” https://github.com/google/proto-quic.

[6] “caddy.” [Online]. Available: https://caddyserver.com