Feasibility Study of MPTCP ACK via Alternative Path in Real Network Environment

Hiroyuki Koibuchi
Graduate School of System and Information Engineering University of Tsukuba
E-mail: koibuchi@osss.cs.tsukuba.ac.jp

Hirotake Abe[*]
Department of Computer Science University of Tsukuba
E-mail: habe@cs.tsukuba.ac.jp

Kazuhiko Kato
Department of Computer Science University of Tsukuba
E-mail: kato@cs.tsukuba.ac.jp

Multi-path TCP (MPTCP) is a network protocol that has the potential to enhance network performance, which is crucial in grid computing systems to improve the overall system's performance. It utilizes multiple network paths between machines to achieve a high network performance. This protocol can also improve the connectivity of multiple servers that may be in different data centers by combining multiple network paths.

One of the main issues of MPTCP is the path for ACK (acknowledgment) packets, which take the same path where original packets come from. When networks have different round trip times (RTTs), MPTCP may dispatch the ACK packet through a longer path, which is inefficient. HayACK solves this issue by selecting the path with the least RTTs. However, HayACK may suffer from inconsistent packets because of middleboxes, which are intermediary devices between the source and the destination that process network packets. These middleboxes may discard MPTCP packets because of inconsistent sequence numbers of HayACK packets, which is a product of HayACK's path selection.

This study aims to investigate the usefulness of HayACK for high-performance data transfer when using multiple paths in real network environments and the influence of middleboxes on HayACK. Our investigation was conducted using various network paths by preparing many client hosts with different IP addresses and one server. The influence of middleboxes is determined by checking the probe packets passed through, changed, or discarded. Assuming the HayACK response, we crafted packets such as TCP SYN packets and TCP ACK packets.

International Symposium on Grids & Clouds 2020, ISGC2020
23rd August - 28th August, 2020
Academia Sinica, Taipei, Taiwan

∗Speaker.

1. Introduction

Recently, cloud computing technology has attracted much attention. Instead of preparing equipment, deploying services and storing data on our own, we can outsource these to an external company and build services on it. The developers do not have to be aware of the outsourced systems and their maintenance requirements. The number of companies adopting cloud computing services compared to on-premise systems has been increasing in recent years.

Typical cloud computing services include Amazon's AWS, Microsoft Azure, and Google's GCP. These service providers have data centers worldwide, and users can use their services without being aware of the country in which the servers are located. The operator does not need to be aware of the physical servers that they handle and can leave the maintenance to external parties. However, if the data centers are in various locations and regions, the operator needs to be aware of the communication between them to maximize the network between these data centers.

Multi-path TCP (MPTCP) is a protocol that can improve communication performance and is an extension of TCP (transport control protocol). MPTCP treats TCP paths as subflows and bundles them together to improve throughput and robustness. An example of the use of MPTCP is the simultaneous use of multiple network interfaces in devices such as smartphones by combining the 4G cellular network and the Wi-Fi network. MPTCP can also combine wired paths that are available among a server and a client.

Morikoshi et al. focused on the path of ACK (acknowledgment) packets for data packets in MPTCP and proposed an alternative path ACK for MPTCP called HayACK [1]. They studied the improvement of throughput in MPTCP by sending ACK packets through a different path than that through which data packets pass. In MPTCP, data packets are sent along the shortest RTT among the TCP routes, and ACK packets, which are replies to the data packets, pass through the same route. HayACK, on the contrary, selects the route with the shortest RTT for the ACK reply. Their results show that the throughput is improved, especially when the difference in RTT between the two routes is significant. The original MPTCP selects the route with the shorter RTT to send data packets and receive ACKs, but only one TCP route is used when the difference in RTT is significant. They concluded that HayACK improves the throughput when the RTT difference between the routes is significant.

However, HayACK is a study based on simulation, which may differ from the real environment. In the real Internet environment, there are devices called *middleboxes*, which are network appliances that relay packets lying between a sender and a receiver. Middleboxes perform operations on packets in addition to relaying packets, such as rewriting or discarding the packet content. Examples of middleboxes are network address translation routers (NATs) and firewalls (FWs). In Honda et al.'s [2] study, they investigated middleboxes on the Internet by sending survey packets over the Internet. They showed that 33% of the packets with inconsistent elements did not pass through. In MPTCP, middleboxes are considered in the design, and inconsistent numbers are adjusted by the protocol side to avoid discarding by the middlebox. In the current design of HayACK, ACK packets are sent to other routes without changing these numbers. Therefore, the middlebox may discard the packet as unreasonable.

This study aims to construct MPTCP with an alternate route ACK, such as HayACK, which can be used in various situations. To propose improvements to HayACK, we investigate the environment in which the current HayACK design cannot work correctly.

In this study, we prepared a route consisting of a single server and a large number of IP addresses for our investigation by referring to Honda et al [2], which points out that HayACK has inconsistent elements in the ACKs of different routes. We constructed survey packets that can mimic the inconsistent elements of HayACK packets and sent survey packets from various IP addresses. To use a large number of IP addresses, we used a VPN (virtual private network) service called VPNGate [3] to conduct the survey instead of using public Wi-Fi spots or home networks. As a result of our investigation, we confirmed that packets with inconsistent sequence numbers and ACK numbers, such as the packets sent by HayACK, passed 82% of the time.

2. Related Work

2. 1 TCP

TCP is one of the network protocols and is located in transport layer 4 of the basic OSI reference model. While UDP (user datagram protocol) is a connectionless protocol that does not require retransmissions, TCP is a connection-based protocol that requires confirmation of arrival. TCP is designed to perform a three-way handshake before establishing a connection.

The packet format of TCP is specified in RFC 793[4]. In the header section of a TCP packet, there is a field called the control flag. These bits can take values of one or zero. For example, when performing a three-way handshake, the client-side first sends a packet with the SYN flag set to one and the ACK flag set to zero. The server sends back a packet with the SYN flag set to one, and the ACK flag is set to one. Finally, the client-side sends a packet with only the ACK flag set to one to establish the connection. There are other flags in the bit area, and Table 1 describes the types of flags handled by TCP.

Among the flags, explicit congestion notification (ECN) is a function of TCP. ECN consists of IP and TCP, and when congestion occurs in the middle of the IP, the TCP congestion controller notifies the congestion or reduces the window.

Table 1: TCP control flags

CWR	Flags for ECNs. Used for notification of congestion window reduction.
ECE	ECN flags. ECN-echo. Used to notify congestion.
URG	Notification of the existence of data for emergency use
PSH	Notify that data will be passed to higher layers
RST	Forcible disconnection of a connection
SYN	Used for a three-way handshake.
FIN	Used to terminate a connection.

Mirja et al. [5] investigated the extent to which ECNs can be used on the Internet. The 2012 results showed that approximately 30% of the 100,000 web servers supported ECNs. The ECN checks whether the ECN flag is available in the initial 3-way handshake. In our study, we sent packets with SYN, ECE (ECN echo) and CWR (congestion window reduced) flags and investigated whether packets with SYN, ACK and ECE flags were returned in the response. In this study, which will be described later, we used the ECN flag of TCP as a reference for the survey packets.

2. 2 MPTCP

MPTCP [6] is an option for TCP extension, which is specified in RFC6824 of type 30 [7]. Originally, TCP could not handle more than two routes, but MPTCP could handle more than two routes. MPTCP uses two or more IP addresses to communicate between the client-side and the server-side and treats each TCP path as a subflow, which can be bundled and treated as a single MPTCP route. An example is a case in which both 4G cellular network and Wi-Fi network are used. This case is shown in Figure 1. In the original TCP, when the TCP communication is changed from 4G to Wi-Fi, it is necessary to reconstruct the TCP communication, but in MPTCP, the communication continues even when one communication method is interrupted.

MPTCP improves communication quality by continuously using two or more TCP paths. In 2017, Apple announced that they implemented MPTCP in Siri, a voice-assistant agent on the iPhone. They reported that this improved the user feedback by approximately 20% and reduced the network failures by a factor of five[1]. When using 4G and Wi-Fi together, there are differences in throughput and stability in each subflow. In MPTCP, there is a mechanism to determine the path that is preferentially used. The default scheduler checks the round trip time (RTT), the arrival time of packets in each path, and sends data packets using the shorter one.

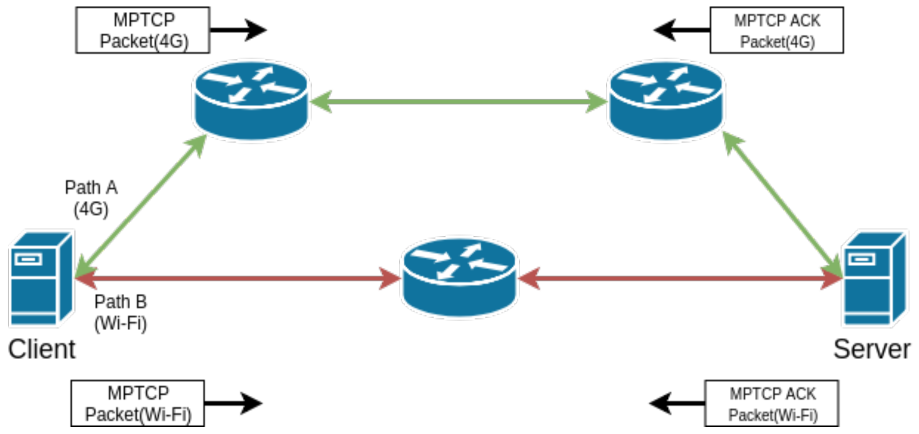

Figure 1: Example of MPTCP using different paths (4G cellular network and Wi-Fi, for instance). Arrows depict flows of packets including both data packets and ACK packets. ACK packets are sent in the reverse direction through the same path that is used when the corresponding data packets are sent through.

[1] Apple. WWDC2017, https://developer.apple.com/videos/play/wwdc2017/707/

2. 3 MPTCP ACK via an alternative path

Morikoshi et al. [1] and Kurosaka et al. [8] improved the ACK of MPTCP by focusing on the ACK path after the data packets of the original MPTCP are sent. First, we describe the existing work called HayACK.

Figure 2 shows a schematic of HayACK. In the original MPTCP, as shown in Fig. 1, the ACKs for a single subflow pass through the same path. However, in the case of HayACK, because the RTT selects the best path at the receiving side of the data packet, it is possible that the ACK will return if an efficient path is selected. Figure 2 shows that the reply to the data packet sent through Path A is returned via Path B. Hence, the difference between the elapsed time RTT of the original MPTCP and the RTT of HayACK leads to improved throughput.

In a study by Kurosaka et al. [8] that focused on ACKs in MPTCP, duplicate ACKs were sent for both paths, that is, for all subflows. Figure 3 shows a schematic diagram of this procedure.

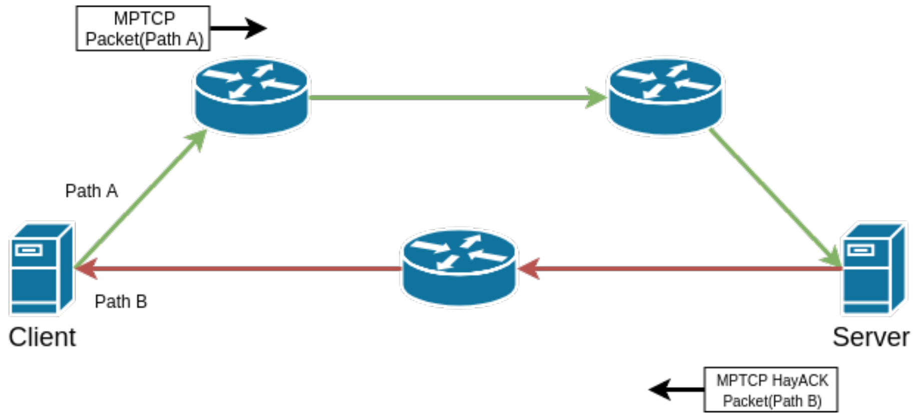

Figure 2: Example of MPTCP with HayACK. Green and red arrows depict the flow of data packets and ACK packets, respectively. Unlike the original MPTCP, HayACK will not always use the same path that the corresponding data packets came through.

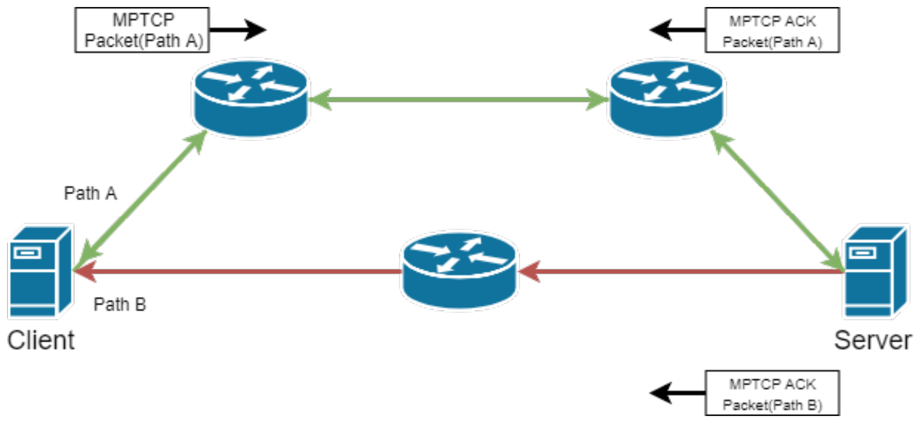

Figure 3: MPTCP used by Kurosaka et al. When necessary, ACK packets are duplicated and send back to the client host through every available path. In these settings, the green path is used for both data and ACK packets and the red path is used only for ACK packets.

In HayACK, only the path with the shortest RTT is selected for transmission. In contrast, in the study by Kurosaka et al. [8], ACKs were transmitted through all available subflows simultaneously. With Kurosaka et al.'s method, duplicate ACKs are sent when packets are transmitted in order, whereas in HayACK, the protocol selects the path with the shortest RTT for all packets. In addition, while both papers use a network simulator called NS-3 [9] for evaluations, HayACK also uses direct code execution (DCE) [10] with NS-3 to ensure that the modified kernel source code can be evaluated using the simulator without reimplementation.

HayACK was evaluated through simulations, including the original MPTCP and the method of Kurosaka et al. It is concluded that the throughput of the MPTCP with HayACK is better than that of the MPTCP with a duplicate ACK, which was used by Kurosaka et al.

2. 4 Investigations of Middleboxes

A middlebox is a device that manipulates traffic, in addition to performing the functions of an IP router. Therefore, in addition to relaying packets, middleboxes may modify or even discard packets in some cases (e.g., FW or NAT). Honda et al. [2] studied whether TCP can be extended in the presence of middleboxes. This may cause packets to be discarded when they are relayed. Specifically, the authors set up a server that receives survey packets and sends survey packets from various locations to the server. The program to send and receive the survey packets is called TCPExposure and is written in Python. The survey was conducted from various IP addresses in various countries, the route from each IP address to the server was considered as a single route, and 142 routes in 24 countries were tested.

The packets were sent to three different destination ports: 80 (http), 443 (https), and 34343 (non-typical port); moreover, TCP extensions (options) were attached to the packets. The authors calculated the percentage of packets that were passed through, changed, or removed, as well as the degree to which packets with TCP extensions may be affected.

In addition, experiments were conducted with a number jump between the sequence number and the number of packets in ACK. When the ACK first arrived, especially for port 80 (http), only 67% of the packets with inconsistent ACK numbers and inconsistent sequence numbers passed through, and the remaining 33% were modified or discarded by the middlebox.

Another previous study has focused on the detection of middleboxes [11]. The authors developed a tool for middlebox detection using ICMP packets, which is called tracebox. Unlike TCPExposure, Traceboxes need to be installed and need to be used only on the sending side.

As shown in Fig. 4, packets sent by HayACK may be discarded by middleboxes because ACKs from other paths may pass through a path with a shorter RTT. These packets appear to have inconsistent sequence numbers because the middleboxes are not aware of HayACK. The original MPTCP adjusts the ACK number and sequence number corresponding to the TCP subflow, and the actual numbers are stored in the payload of MPTCP to ensure that it can pass through middleboxes. However, HayACK is not designed to adjust the numbers, because of the processing time required for the adjustment and the possible problems caused by the adjustment.

Morikoshi et al. [1] performed an evaluation as a simulation and assumed that middleboxes do not discard packets. Therefore, the ACK number or sequence number was not adjusted when replying.

Thus, it is necessary to verify how the system is affected in a real environment and the number of packets that are discarded.

The packets that need to be tested are those with inconsistent sequence numbers or ACK numbers, as described by Honda et al. [2]. Packets sent from HayACK-enabled servers might be regarded as inconsistent packets to a middlebox that considers packets on a single path. The survey results may be different in the existing Internet, and the middleboxes that exist may have changed. In addition, the results of the survey may be influenced by the sending of duplicate ACKs in the MPTCP [8].

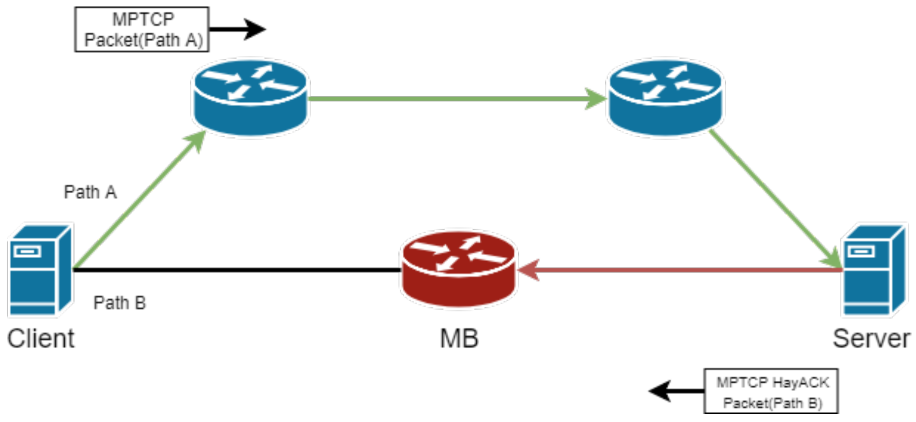

Figure 4: HayACK packets discarded by a middlebox. Because the middlebox in red does not know the existence of green flow, it cannot determine that an ACK packet from the server is legitimate or not. Some middleboxes will discard such packets.

2. 5 Investigation Environment

In the study by Honda et al., packets were sent through 142 routes from various countries to a single server. Other wide-area testbeds are also available on the Internet, such as PlanetLab[12]. These testbeds allow for conducting network experiments in environments similar to the Internet. In the present study, we used a VPN service called VPNGate to conduct our research on the Internet.

VPNGate is a volunteer-based VPN service proposed by Nobori et al. [3]. The service is mainly intended to allow users to freely access information on the Internet bypassing controls enforced by a firewall device from countries or regions that are not otherwise accessible. By connecting to VPN servers distributed worldwide, users can access sites that are inaccessible from their own area, change their IP address when they access such sites, and protect themselves from eavesdropping on free Wi-Fi using VPN.

VPNGate provides client software for various operating systems. With the software, a user can be a volunteer who provides VPN servers to other users, as well as using VPN servers provided by other users.

3. Investigation Method

3. 1 Method

We conducted several investigations to find out how the HayACK can perform as intended in the real Internet environment where various middleboxes are deployed. The basic idea of the investigation is the same as Honda et al.'s[2]. We set up a server with a single global IP address and generate various patterns of TCP data flows from many clients. Even though the server is physically placed in our laboratory at University of Tsukuba, it is not connected to its campus network but connected to another network that is operated by a different network operator (open.ad.jp). We conducted the following two types of investigations:

One is “investigation using spontaneous packets." This is a preliminary investigation to confirm the existence of the middlebox and prepare for the main investigation described later. In TCP, 3-way handshaking is performed at the beginning, and packets are exchanged. However, middleboxes discard or change the contents of various packets in addition to packets with inconsistent numbers. In this study, we sent a survey packet only once without any connection, to confirm the existence of the middlebox. Unlike in 3-way handshaking, the client-side sends an arbitrary packet only once, and the server-side only receives the packet, without replying.

The other is "an investigation using mimicked connection." This is a verification method that involves sending an investigation packet mimicking the alternate-path ACK of HayACK. In TCP, a TCP connection is established by 3-way handshaking, and then data exchange begins. The sender of a data packet receives an ACK in reply from the destination. Each packet contains a sequence number and an ACK number corresponding to the amount of sent data. Therefore, if these numbers are incorrect, the middlebox may discard the packet as a suspicious packet. This investigation was conducted as the main study.

Figures 5 and Figure 6 show the cases of one-to-one and one-to-many investigations, respectively. We explain the cases of making a connection and sending an unexpected ACK.

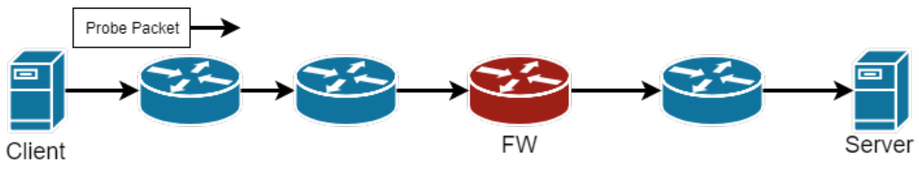

Figure 5: One-to-one investigation when sending spontaneous packets. ACK packets which are not corresponding to any existing TCP connections are sent from a client in typical locations (e.g. a campus network, a public WiFi in a city, etc.) to the server. If a FW or another middlebox which discards such packets exists in the middle of the path, the server will not receive the ACK packets.

Figure 5 shows a one-to-one study of sending an unexpected packet. As shown in the figure, we assume a middlebox between the client and server. When sending an unexpected packet, the server must be ready to receive the packet first. To mimic a connection, it is necessary to prepare a connection in advance instead of simply sending a one-way survey packet, as shown in Figure 5. After setting up the connection, we sent a survey packet that imitated the ACK of another route.

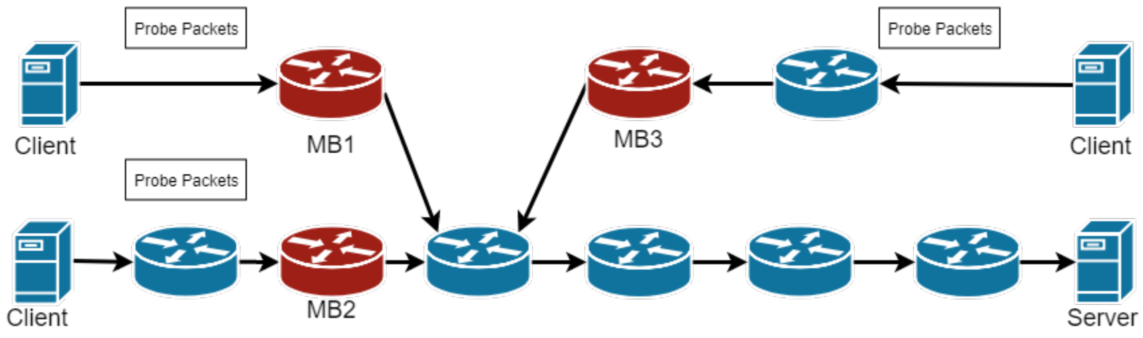

Figure 6: One-to-many investigation when sending spontaneous packets. Unlike the one-to-one investigation, we use randomly selected clients and collect as many results as possible to draw statistical characteristics.

Figure 6 shows a one-to-many survey when an unexpected packet is sent. We performed a one-to-one investigation on various routes. Just as there were middleboxes in between in the one-to-many case, there could be different middleboxes for each route. We expect that some of the middleboxes are the same and that some of them behave differently.

In the case of replicating the connection, it is necessary to connect each route first. The process of setting up a connection and sending an ACK mimic packet to another route is the same as that for the one-to-one case.

We used Python and Scapy for the experimental environment and HayACK packets. Scapy [13] is a tool that can analyze captured packets and create its own packets. It works with python2 and python3 on Linux and many other unix-like platforms. It is also available as a Python library and can send crafted packets to a specific server.

3. 2 Investigation using spontaneous packets

The investigation using spontaneous packets was conducted as follows. The client-side has the function of constructing and sending packets using Scapy. However, because it sends packets unilaterally, we do not provide a receiving function. For the packets to be sent, we focus on the flag and port numbers. The client-side must start communication in the initial 3-way handshaking to ensure sending anything other than SYN packets does not start communication. In Scapy, it is possible to set the bit of any control flag to 1. In this preliminary study, we used Scapy to send packets to the server side without any packet connections with flags other than SYN packets. The server-side only captured packets sent by Scapy. Scapy captured all packets arriving at the NIC and could filter the packets by port number.

Table 2 lists the types of packets used in the investigation. In particular, we focused on ports and flags. For the port, we use either 80, 443, or 34343, and for the flags, we used SYN, SYN+ECE, SYN+ACK, ACK+ECE, and ACK. In this case, a total of 15 packets were sent per route because the spontaneous packets used in the study were a combination of these.

Table 2: Values set to the header fields of packets used in the investigation with spontaneous packets

protocol	field	options for value
IP	src	IP address of the client
IP	dst	IP address of the server
TCP	sport	Any (random)
TCP	dport	80(http), 443(https), 34343(non-typical)
TCP	seq	Any (random)
TCP	ack	Any (random)
TCP	flags	SYN, SYN+ECE, SYN+ACK, ACK+ECE, ACK

3. 3 Investigation using mimicked connections

The investigation using mimicked connections was conducted as follows. In addition to the functions of packet construction and transmission, the client-side performs the function of reception. To create a TCP connection, it is also necessary to construct and send packets based on the reply packets from the server. In addition to the packet-capture function, the server-side performs a socket communication function for the connection. We explain the implementation of 3-way handshaking using Scapy. Normally, when we establish TCP communication from Python, we can use a socket API provided by an underlying operating system, and we need not be aware of 3-way handshaking. Because Scapy can create, receive, and send any form of a packet, it is possible to implement 3-way handshaking without the help of operating systems.

After a connection is established, data packets and ACK packets, which are replies to the data packets, are repeated. The survey packet is sent after the connection is established. Table 3 shows the actual packets used. In addition to the port number and flag, we focused on the sequence number and ACK number.

Table 3: Values set to the header fields of packets used in the investigation with mimicked connections

protocol	field	options for value
IP	src	IP address of the client
IP	dst	IP address of the server
TCP	sport	Any (random)
TCP	dport	80(http)
TCP	seq	Sequence numbers containing inconsistencies
TCP	ack	ACK number containing inconsistency
TCP	flags	ACK, ACK+ECE, SYN+ACK, ACK+CWR, or ACK+ECE+CWR

Table 4 shows the breakdown of the packets, including 3-way handshaking. In addition, we start sending packets from the client to the server on port 80. We used the constants for client_num and server_num and set the number from the client to 5500. The sequence numbers from the server and server are random. The sequence number from the server, server, is random and is written as +1 for each additional number. The data packet consists of five bytes of "hello." Therefore, the ACK number is added as +5 to the reply of the data packet. The survey packets should be six or more. To create inconsistent sequence numbers and ACK numbers, we add 50 to each fourth packet in this survey and send it.

Table 4: Overview of 3-way handshaking packets. “server_num” and “client_num” correspond to initial values.

Order	Direction	Packet Type (flag)	Seq. Number	Ack. Number
1st	Client-Server	SYN	client_num	0
2nd	Server-Client	SYN+ACK	server_num	client_num+1
3rd	Client-Server	ACK	client_num+1	server_num+1
4th	Client-Server	Data(PSH+ACK)	client_num+1	server_num+1
5th	Server-Client	ACK	server_num+1	client_num+1+5 (5 corresponds to the size of data in 4th packet)
6th	Client-Server	Probe packets	client_num+1+50 (50 is a random offset chosen to cause inconsistency)	server_num+1+50

4. Investigation Result

4. 1 Result: Investigation using spontaneous packets

In this experiment, we used VPNGate [3] and assumed that packets were sent from various sources and countries. As in the first experiment, we consider sending spontaneous packets to the server. We used Windows 10 on the sender side and Ubuntu 18.10 on the receiver side. Figure 7 shows a schematic diagram.

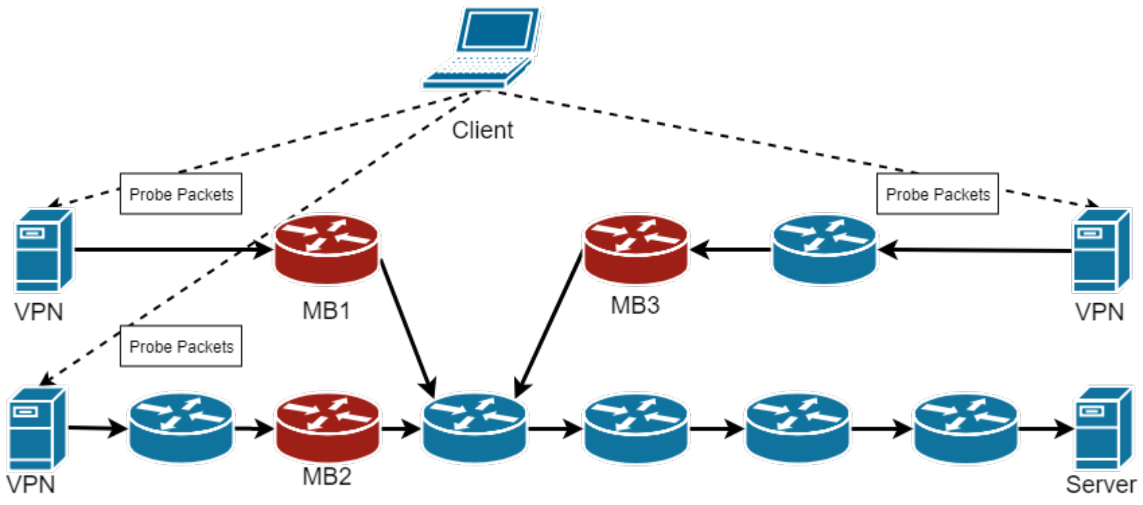

Figure 7: Schematic of an experiment using the VPNGate. The client establishes VPN connections and sends probe packets through the connections. For each trial, the client selects a new VPN server from the available server list posted on the VPNGate site.

The survey was conducted from our laboratory through a campus network of The University of Tsukuba, Japan. Table 5 is a recapitulation of the results. Spontaneous packets sent with SYN or SYN+ECE flags are successfully received by the server. All the spontaneous packets from the lab did not reach, except for SYN and SYN+ECE, and SYN+ACK, ACK+ECE, and ACK did not reach all ports. The selection of the destination port number does not affect the result. Note that the server is connected to a different network. The result indicates that there are a middlebox in between the path between the client and the server and it discarded the packets with specific flags.

Table 5: Recapitulation of the results of the survey where the lab sent out spontaneous packets

	SYN	SYN+ECE	SYN+ACK	ACK+ECE	ACK
Port 80 (http)	pass	pass	lost	lost	lost
Port 443 (https)	pass	pass	lost	lost	lost
Port 34343 (non-typical)	pass	Pass	lost	lost	lost

Table 6: Results of an investigation using VPNGate to send spontaneous packets. IP means the number of IP addresses used in this investigation. S, A, and E stand for SYN, ACK, and ECE flags, respectively. SE means SYN+ECE. Numbers after the flags mean the port number. The country name is in ISO codes.

country	IP	S:80	S:443	S:34343	SE all	SA all	AE all	A all
jp	73	73	73	73	14	0	12	12
kr	76	76	76	76	4	0	0	0
us	6	6	6	6	0	0	0	0
hk	1	1	1	1	0	0	0	0
id	1	1	1	1	0	0	0	0
ve	1	1	1	1	0	0	0	0
vn	1	1	1	1	0	0	0	0
my	1	1	1	1	0	0	0	0
ph	1	1	1	1	0	0	0	0
ro	1	1	1	1	0	0	0	0
th	1	1	1	1	0	0	0	0
SUM	163	163	163	163	18	0	12	12
%	100	100	100	100	11	0	7	7

Table 6 shows the results with VPNGate. There are some servers whose VPN connections were rejected or failed,which are excluded from the results and do not affect the ratio of successful trials.

ACK+ECE and ACK packets were found to pass through the path. This is an example of packets that did not pass, even when sent from the laboratory. This indicates that the packets that could not pass through the VPN server before connecting to the VPN server could pass through the VPN server. In other words, using the VPN server, it is possible to pass the packets that do not pass through the alternate-route ACK. In the case where SYN+ECE packets passed and the case where SYN+ECE, ACK+ECE, and ACK packets passed, only SYN+ECE packets passed, and ACK+ECE and ACK packets did not pass because the middlebox discarded them after connecting to the VPN server.

On the other hand, VPNGate was not able to handle SYN+ECE packets when we started this investigation. We reported this issue to its author, which was fixed in 2020 [14]. Older versions of VPNGate judge TCP packets with only the SYN flag are set as valid SYN packets and discard SYN packets with any other flag. Therefore, with the older versions, SYN+ECE packets were never considered correct SYN packets, resulting in unexpected losses regardless of the path between a VPNGate server and the server. The above results can be attributed to the gradual increase in the number of VPNGate servers that have been modified.

4. 2 Result: Investigation using mimicked connections

We used VPNGate to mimic the connection. As shown in Figure 8, we sent a survey packet to a TCP connection. In this study, we used Ubuntu 18.10 on both the sender and receiver sides.

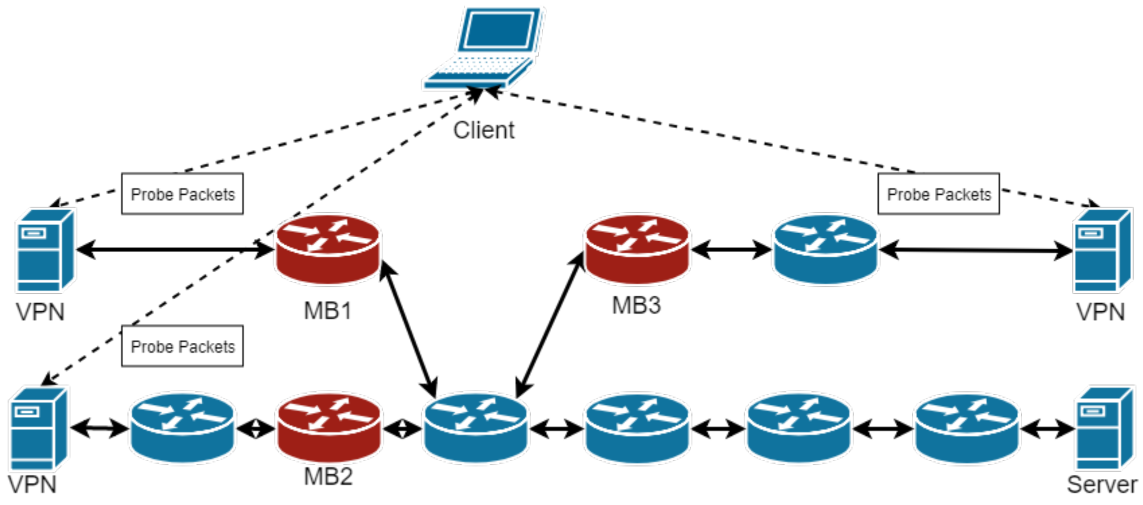

Figure 8: Connection made using VPNGate. It is mostly the same as Figure 7, excepting the direction of communications. The communication is bi-directional to establish a TCP connection between the server and the client.

The results in Table 7 were obtained after establishing the connections. When no connection is established, the ACK and ACK+ECE packets from the laboratory do not pass, but in this case, they do. In addition for ECN,we sent packets with CWR and ECE+CWR added and confirmed that both packets were passed through.

Table 7: Results sent from the lab on port 80

	ACK	ACK+ECE	ACK+CWR	ACK+ECE+CWR
Port 80	pass	pass	pass	pass

Table 8: Results of the investigation of unreasonable ACK packet passing on port 80. IP means the number of IP addresses used in this investigation. A, E, S, and C stand for ACK, ECE, SYN, and CWR flags, respectively. AE means ACK+ECE.

country	IP	A：80	AE:80	SA：80	AC：80	AEC:80
KR	44	43	43	0	43	43
JP	31	21	21	0	21	21
VN	3	2	2	0	2	2
RS	2	1	1	0	1	1
AR	1	1	1	0	1	1
IN	1	1	1	0	1	1
CL	1	1	1	0	1	1
NO	1	0	0	0	0	0
TH	1	0	0	0	0	0
KH	1	0	0	0	0	0
LV	1	1	1	0	1	1
SUM	87	71	71	0	71	71
%	100	82	82	0	82	82

We sent SYN+ACK packets to 11 countries on 87 routes and used VPNGate. Table 8 shows the the results. It was confirmed that 82% of the packets with inconsistent sequence numbers and ACK numbers, such as HayACK, passed through.

4. 3 Summary and Discussion

The first thing we can learn from the results is that the acceptance rate of ACK packets with inconsistent sequence numbers is getting better in the last decade. At the time that the investigation by Honda et al. [2] was conducted in 2011, it was 67%. Although it is not the completely same setting as Honda et al., the result of our investigation was 82%. This suggests that chance that newer protocols that use communication de facto standards on the Internet in irregular ways such as HayACK are getting more chances to work in the real environment.

Another thing we can learn is that mimicking 3-way handshaking can improve the acceptance rate of inconsistent ACK packets. Without the mimicking, the acceptance rate of inconsistent ACK packets was 67%. With the mimicking, the acceptance rate becomes 82%. This suggests that mimicking 3-way handshaking is still a good way to deploy newer protocols with inconsistent ACK packets.

We must also mention that ECN flags (ECE, EWR) flags do not improve the acceptance rate of ACK packets with inconsistent sequence numbers. Rather, with our experience with older VPNGate implementations that always discards SYN packets with any other flags such as ECE, ECN flags can be a cause of making the acceptance rate worse because the implementation related to ECN might be well tested comparing to normal TCP implementations, which are under load testing by billions of people in the world.

5. Conclusion

Improving existing communication protocol standards such as TCP sometimes faces a problem that even a slight deviation from standards can cause an unexpected loss of such packets in the middle of the path from a sender to a receiver. In this study, we investigated how TCP packets with inconsistent sequence numbers, which can be caused by a modified version of MPTCP, can work in the current real network environment.

We conducted the investigations using a VPN service to test various communication paths. Our results show that the acceptance rate of such inconsistent packets was 82%, which is better than the results in 2011 reported in Honda et al. [2]. This suggests that we have more chances to re-design existing communication standards such as TCP.

The assumed cause of such losses of irregular ACK packets is the existence of middleboxes in a communication path. It can be an intended operation designed by an operator who deployed it. However, the case we experienced with VPNGate, which is caused by the wrong disposal of packets with irregular flags, suggests that there are cases that such irregular packets are dropped unintentionally. Our future work includes further investigation of the distribution of middleboxes in the real network environment and the cause of such disposals.

References

[1] Yusuke Morikoshi, Hirotake Abe, and Kazuhiko Kato. HayACK: exploiting characteristically diverse paths to achieve quick ACKing in MPTCP. International Workshop on the Future of Cloud Computing and Cloud Services (FutureCloud 2017, held in conjunction with IEEE CloudCom 2017), 8 pages. Hong Kong, China, 2017.

[2] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. 2011. Is it still possible to extend TCP?. In Proceedings of the ACM SIGCOMM conference on Internet measurement conference (IMC '11), 2011.

[3] Daiyuu Nobori and Yasushi Shinjo. VPN gate: a volunteer-organized public VPN relay system with blocking resistance for bypassing government censorship firewalls. In Proceedings of the 11th USENIX conference on Networked Systems Design and Implementation (NSDI'14). USENIX Association, Berkeley, CA, USA, 229-241, 2014.

[4] J. Postel (ed), Transmission Control Protocol, RFC793, available at https://datatracker.ietf.org/doc/html/rfc793 [accessed 20 April, 2021]

[5] Mirja Kühlewind, Sebastian Neuner, and Brian Trammell. On the state of ECN and TCP options on the Internet. In Proceedings of the 14th international conference on Passive and Active Measurement (PAM'13), Matthew Roughan and Rocky Chang (Eds.). Springer-Verlag, Berlin, Heidelberg, 135-144, 2013.

[6] C. Paasch and O. Bonaventure., Multipath TCP, Communications of the ACM, 57(4):51-57. April 2014.

[7] A. Ford et al., TCP Extensions for Multipath Operation with Multiple Addresses, RFC6824, available at https://tools.ietf.org/html/rfc6824. [accessed 20 April, 2021]

[8] Takumi Kurosaka and Masaki BandaiMultipath TCP for heterogeneous environments. Online ISSN 2187-0136. IEICE ComEX, Volume 4, no. 6, 211-216, 2015.

[9] Riley G.F., Henderson T.R. (2010) The ns-3 Network Simulator. In: Wehrle K., Güneş M., Gross J. (eds) Modeling and Tools for Network Simulation. Springer, Berlin, Heidelberg.

[10] Hajime Tazaki, Frédéric Uarbani, Emilio Mancini, Mathieu Lacage, Daniel Camara, Thierry Turletti, and Walid DabbousDirect code execution: revisiting library OS architecture for reproducible network experiments. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies (CoNEXT '13), 217-228, 2013.

[11] Gregory Detal, Benjamin Hesmans, Olivier Bonaventure, Yves Vanaubel, and Benoit Donnet. 2013. Revealing middlebox interference with tracebox. In Proceedings of the 2013 conference on Internet measurement conference (IMC '13). ACM, New York, NY, USA, 1-8

[12] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. 33, 3 (July 2003), 3–12, 2003.

[13] Scapy - Packet crafting for Python2 and Python3, available at https://scapy.net/ [accessed 20 April, 2021]

[14] VPNGate modify history, https://ja.softether.org/5-download/history [accessed 20 April, 2021]