Measuring transport layer evolvability in the Internet

Yukai Xiao

Master of Science

School of Informatics

University of Edinburgh

2024

Abstract

Evolvability is often an beneficial design principle when technology is created, and transport layer protocols such as TCP and QUIC follow this principle. However, with the diversification and complexity of network infrastructure, the deployment of new extensions or new protocols may be affected by middleboxes. Does the current Internet allow transport layer protocol optimization? To investigate this question, we constructed packets that simulate future protocols and observed potential behaviors of middleboxes using active measurement. Our goal is to provide valuable reference information for protocol designers and internet service providers (ISPs).

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yukai Xiao*)

Acknowledgements

I would like to thank my supervisor Michio Honda for giving me the chance to do this project. His insightful guidance is really helpful for my dissertation.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Transport layer protocols are very important in the Internet, it is mainly responsible for establishing a communication channel between the ports of two hosts. In the development of the modern communication architecture, in order to meet the requirements for performance, privacy protection, and stable transmission, the protocols within it have experienced rapid iteration. From UDP, which does not require formal connection establishment, to TCP, which is a highly reliable connection-oriented protocol, to QUIC, which aims to overcome the limitations of TCP, the transport layer has been significantly enhanced [22].

These protocols have been designed with the possibility of future expansion in mind [1]. However, with the introduction of a large number of middleboxes (such as firewalls, NATs, load balancers, and proxy servers) in the network path, the brand new protocols and the new capabilities of known protocols face the challenge of not being widely deployed [18]. These devices play an important role in enhancing network security and performance. But at the same time, they make the earlier principle of end-to-end transparency discrediting, as they can prevent new protocols or packets with new extensions from going through, or force those packets to fall back to their original versions. All of these behaviors limit the further development of the transport layer and affect the enthusiasm of technicians to develop new protocols. Therefore, the innovation and evolvability of transport layer protocols in the unfamiliar network path becomes the issue that the project wants to discuss.

However, there are few studies on the tolerance of the existing Internet infrastructure to new extensions or new protocols. Past research has shown that while theoretically

feasible, practical deployments face various effects caused by middleboxes behavior [18]. These effects indicates the necessity of in-depth measurement and summary of transport layer evolvability, so as to provide data support and design direction for designers of network architecture to adjust transport layer protocols.

1.2 Objectives

This project will explore the evolvability of transport layer protocols in the 2024 Internet. The main goal is to create a simulated future TCP and QUIC packets, and to analyze the results of these traffic in a network environment containing middleboxes through active measurement methods. Specific objectives are as follows:

1. Implement tools for generating TCP traffic that simulates future protocol extensions.
2. Implement a tool for generating UDP traffic that emulates the QUIC protocol.
3. Tools are respectively deployed on the client side and the server side for packet generation and response packets.
4. Use active measurement to measure the impact of middleboxes on protocols containing unknown extensions, including whether the unknown extension is removed, whether the packets are allowed to pass through, whether the initial sequence number is modified, and how the retransmitted packets behave when the inconsistency occurs.
5. According to the testing results, the suggestions to keep transport layer evolvability can be proposed.

1.3 Structure of Dissertation

The first chapter is about motivation and objectives of our study. In the second chapter, the backgorund knowledge of transport layer protocols and middleboxes are introduce, and it also contains related work of this project. After that, the implementation of tools used and testing environment are explained in the third chapter. Then, the results obtained from five types of tests can be found at the fourth chapter. In the final chapter, we summarized the research results. At the same time, we acknowledge current limitations and point out the direction of future efforts.

Chapter 2

Background

2.1 Importance of transport layer protocols

In modern networks, transport layer protocols such as TCP and QUIC play a central role, responsible for achieving reliable data transmission between both ends of the network. TCP ensures the correct transmission of data through its complex error detection, acknowledgement retransmission and flow control mechanism. As a new UDP-based protocol, QUIC aims to reduce the delay of connection and transmission, and provide the same reliability as TCP [19]. Because QUIC integrates many security and performance improvements in design, it is gradually accepted by the industry, especially in real-time applications and mobile networks [16].

2.2 Evolution of TCP and QUIC

The purpose of introducing TCP protocol is to provide reliable data transmission and ensure the order and integrity of data packets. It uses the three-way handshake mechanism to establish the connection channel, and uses the acknowledgement and retransmission mechanism to ensure reliable transmission. With the development of the Internet, TCP introduces flow control and congestion control to deal with the dynamic changes of network resources. To further improve performance and compatibility, TCP gradually introduces more extensions, such as Selective Acknowledgement (SACK) [7], and TCP Fast Open [23].

The QUIC protocol has experienced significant updates since it was produced by Google in 2012 [24]. In order to achieve wider application, IETF has standardized it, making QUIC more versatile [13]. In the standardization process, the version

negotiation mechanism has been added to enable the client and server to negotiate which QUIC version to use when connecting for the first time, allowing the existing communication not to be interrupted when updating the version. In addition, in order to comply with a broader network security standard, QUIC uses TLS 1.3 as the encryption layer, which is different from the encryption mechanism in the original version [16].

2.3 Middleboxes

In the operation of these protocols, middleboxes such as firewalls, NAT devices, load balancers and proxy servers play an important role. These devices are designed to monitor, control and optimize network traffic to improve network security and reliability [27]. For example, the firewall is responsible for checking the data packets entering and leaving the network, and preventing the communication that is unsafe or does not conform to the rules; NAT devices solve the problem of IPv4 address exhaustion. By mapping multiple private IP addresses to one or several public IP addresses, multiple devices can share a public network connection; The load balancer improves the response speed and the overall fault tolerance of the system by distributing requests from clients to multiple servers; The proxy server acts as an intermediary between the client and the server, providing data caching services to reduce latency, while performing user authentication and content filtering.

However, they may also have some negative effects. First, the middle boxes may destroy the end-to-end transparency of the network, because they may modify or intercept the data packets passing through them [8]. This behavior may conflict with the featrues of new protocols, especially those designed to improve network performance and security. For example, some extended features of TCP, such as Explicit Congestion Notification(ECN) and Multipath TCP, may not work properly due to the intervention of the intermediate box [18].

2.4 Related Work

2.4.1 TCP

There have been many studies on the interaction between the protocol and the middleboxes. Medina et al. [18] discussed how these devices affect end-to-end network communication. They proposed a measurement tool called TBIT, which is used to

detect the behavior of various transmission protocols in the network, especially TCP protocol. This article focuses on ECN, Path MTU Discovery (PMTUD), and TCP options. The results show that middleboxes can affect TCP performance in various ways. For example, some middleboxes block TCP SYN packets trying to negotiate ECNs, and some devices fail to return necessary ICMP messages, leading to PMTUD failure. These results prove that as early as 20 years ago, there was a difference between the actual Internet and the Internet described in theory.

The work of Michio et al. [10] evaluated the impact of middleboxes on TCP expansion, especially how these devices handle the newly introduced TCP options. Using a measurement tool called TCPExposure, the author recorded the behavior of middleboxes from 142 networks around the world. They not only tested whether the network path allowed the transmission of unknown TCP options, but also included sequence number modification test, sequence space holes test and packet retransmission test. Finally, the paper specifically discusses the impact of these results on Multipath TCP (MPTCP), TcpCrypt, and extended TCP option space. And it gives some suggestions on how to design these extensions to work effectively in the environment where middleboxes exist. The active measurement method used by the author has greatly inspired the project. This method can effectively observe the behaviors of middleboxes in the path by deploying the sender and receiver tools between two hosts.

Kühlewind et al. [15] focused on the actual deployment of ECN in the Internet, which is a technology designed to notify the sender when the network is congested, not by packet loss but by marking packets. They also used active measurement and passive measurement. The results show that ECN support rate in IPv6 networks has reached 47.52%, while IPv4 support rate is only 29.48%. At the same time, many ECN capable connections do not actually use ECN for congestion control. It is worth noting that although the deployment of ECN is increasing, its actual utilization rate in the network is still lower than other TCP options such as SACK, Timestamp (TS) and Window Scaling (WS) due to the interference of network intermediate devices. In this project, we will not focus on the deployment of a known option, but on those completely unknown extensions.

The MBtest proposed from [9] is a set of Click element based models used to simulate the behavior of middleboxes. These elements can represent various interferences that may exist between TCP extensions and intermediate devices. Through these elements, researchers can experimentally evaluate the response of TCP to various types of intermediate devices in a controlled environment. The options tested in this

article include SACK, TS, large window support, and MPTCP. The results show that Middleboxes may segment or merge data segments during transmission, which is particularly challenging for MPTCP that relies on precise data order and size. In addition to conducting simulation experiments using MBTest, the author also deployed MPTCP in 50 different networks and observed that some devices did not handle MPTCP correctly. MBTest can precisely control the behavior of middleboxes, making it very suitable for testing in laboratory environments. However, our project needs to collect data from real network.

Craven et al. [5] developed a system called TCP HICCUPS, which is a new TCP extension used to detect packet header modifications that may occur on the transmission path between the two ends of a TCP connection. This system enables end users and network applications to better understand the behavioral changes of their data during transmission over the network, especially how it is processed and modified by middleboxes. After possessing this ability, data packets can adaptively adjust their behavior in response to modifications made to middleboxes. They deployed and tested on thousands of different paths to highlight the path transparency and network diagnostic capabilities brought by HICCUPS.

Edeline and Donnet [6] argue that despite the general assumption of intermediate boxes, there is very limited data on their actual deployment, particularly on how autonomous systems (AS) deploy them for universality and persistence. Their experiment used Tracebox to identify modifications made by Middlebox to network traffic. The results show that middlebxoes are not as common as standard network devices, and most are deployed at the boundary of AS. Meanwhile, once deployed, middleboxes will continue to run without requiring too many changes.

2.4.2 QUIC

With QUIC first cited by Google in 2012, there have been some studies in recent years focusing on the interaction between QUIC and middleboxes. From its own characteristics, this protocol encrypts both the header and payload, which prevents intermediate devices from checking and processing the content of data packets, posing a challenge for devices that rely on packet inspection to perform their functions [3].

Chaudhary et al. [2] investigated the impact of midboxes interference causing QUIC to fall back to TCP on the quality of experience of YouTube video streaming. They set two different configurations in the Chrome browser, one to enable the QUIC protocol

and the other to disable QUIC (i.e. using TCP). And tools were used to control the network bandwidth and other network parameters in the testing environment, allowing researchers to compare the performance of the two protocols under the same network and video conditions. After collecting over 2600 hours of YouTube video streaming data, they found that in over 60% of cases, traditional TCP settings either performed better than browsers with QUIC enabled or performed the same. This means that QUIC may not always provide better performance or experience. This may be because QUIC needs to fallback to TCP when encountering network intermediate devices blocking its UDP packets, which affects performance and video stream quality. Therefore, these findings suggest that QUIC designers need to reconsider their fallback strategies to better handle intermediate device blocking and network instability.

Kosek et al. [14] developed an enhanced version of QUIC called Secure Middlebox-Assisted QUIC (SMAQ). QUIC uses a traditional end-to-end encryption model to prevent intermediate boxes from accessing or modifying transmitted data. This may limit the functionality of the middle box, which may be required for network management or performance enhancement. Therefore, the SMAQ model allows for selective exposure of connection and protocol information to the intermediate box. This method enables endpoints to consciously insert intermediate boxes into QUIC connections while maintaining end-to-end encryption, and control which information is shared with the intermediate boxes. The SMAQ protocol includes additional security layers, which ensure the confidentiality and integrity of data even when middleboxes actively modify traffic. They conducted a detailed evaluation of SMAQ in a distributed performance enhanced agent (PEP) environment, and the evaluation showed that SMAQ can significantly improve performance, especially in high latency and packet loss scenarios.

Chapter 3

Methodology

3.1 Dependency

The construction, sending and sniffing of data packets in the project mainly depend on Scapy [26]. This is a python-based network packet operation tool. It supports a wide range of network protocols and can flexibly create and customize packets according to the needs of the project. In this project, the goal is to explore the evolvability of TCP and QUIC by generating data packets that may appear in the future. The function of Scapy is critical to the realization of this goal. The reasons can be summarized as follows:

1. Scapy allows you to customize each field of a packet to generate a packet with specific modifications, such as adding unknown options to the TCP header and fields in the QUIC protocol to the UDP payload. This precise packet construction capability allows us to test how the middle box processes these data in specific scenarios.

2. Scapy can capture and analyze the response in the network, so that we can determine whether the data is discarded or modified in the network path.

3. By sending and receiving these customized data packets, we can actively measure and analyze the behavior of the network and its middleware.

4. Scapy has an active development community and complete documentation, which helps with code quality and development speed.

There are many tools similar to Scapy, such as hping [25], Libcrafter [21], Ostinato [20], WireShark/TShark [28]. However, compared to Scapy, hping is a TCP/IP packet

assembly and analysis tool that can only be run on the command line, and it lacks protocol support and flexibility in packet construction compared to Scapy; Libcrafter is a network packet construction library developed using C++. Although it has powerful performance, its scripting ability is worse compared to Python based Scapy; Obstinao provides a graphical interface, so it cannot run on remote servers and requires payment for use; TShark is the command-line version of WireShark, which is powerful in packet capture and inspection, but far inferior to Scapy in packet construction and manipulation.

In summary, Scapy integrates the construction, sending, receiving, and analysis of data packets, with comprehensive functionality and the ability to precisely customize data packets at various levels of the network stack that other tools cannot achieve. Meanwhile, as a Python library, Scapy can easily integrate with other python tools, providing more automation and adapting to more complex testing environments.

3.2 Active Measurement

In this project, active measurement refers to actively detecting network behavior by sending customized data packets (such as TCP SYN packets) to evaluate the functionality and characteristics of a certain protocol, as well as the impact of network intermediate devices on transmission protocols.

Based on the requirements and objectives of this project, proactive measurement has the following advantages:

1. By actively sending data packets, experimenters can directly and clearly understand how the intermediate and how it affects the functionality of TCP extensions and QUIC protocols.

2. Active measurement can reveal some behaviors that are difficult to detect by passive measurement. For example, by sending SYN packets with options, hidden behavior in the network path can be revealed. If the intermediate box in the network blocks or modifies these packets, it will cause the client to be unable to connect or successfully create a connection but its extension will fail.

3. Active measurement can provide real-time analysis of received responses, thereby providing more detailed network behavior data.

The opposite of active measurement is passive measurement, which is defined as analyzing existing network traffic data to evaluate the actual usage of a protocol. There are three differences between these two measurement methods:

1. **Measurement range**: Active measurement is mainly used to evaluate the functionality of networks or servers and determine the support for certain protocol features; Passive measurement is used to analyze the actual deployment of protocols and the characteristics of network traffic.

2. **Data collection method**: Active measurement obtains responses by sending special data packets to the network, while passive measurement analyzes existing network traffic for statistical purposes.

3. **Usage scenario**: Active measurement is suitable for verifying the behavior of intermediate boxes and the adaptability of protocol extension functions, while passive measurement is used in evaluating the deployment of protocols and analyzing traffic patterns.

Considering the principle of end-to-end transparency in the early design of the Internet, the project should use active measurement to detect and analyze the behaviors that can occur in the network path.

3.3 Testing Environment

3.3.1 Client - Sever Model

The experiments conducted in this project adopted a client server model (also known as a sender receiver model). The client actively sends TCP or QUIC packets to the server, and the server will selectively listen for packets from the client's IP address and issue different responses based on the different packets. In all the experiments conducted, the client used the author's personal computer with an operating system of MacOS and a system version of `MacOS Monterey 12.7.2`.

The server uses Cloudlab, which is a cloud experiment platform that provides infrastructure and services for scientific research and education. Conducting experiments on this platform can eliminate the impact of the server on the data packet. If the measurement results show that the data packet has been modified, it can be determined that it is caused by the intermediate box. In addition, Python and Scapy are installed on both the client and server, with the former version being `3.9.7` and the latter using version `2.5.0`.

3.3.2 Block RST

The TCP stack is one of the core components in network communication and a part of the operating system kernel, responsible for managing the establishment, maintenance, and termination of TCP connections. Through its provided API, developers can easily establish TCP and send and receive data without dealing with the underlying network transmission details. For example, when an application calls `connect()`, the TCP stack is responsible for performing a three-way handshake and establishing a connection.

When a SYN packet is directly sent from the client to the server, the server-side Scapy script may encounter a situation where it cannot listen to the SYN packet and the client will receive an RST packet sent from the server. After research, it can be found that this RST packet comes from the TCP stack on the server side. Due to the fact that the Scapy script that listens for packets is only running on the server, and the destination port of the SYN packet is closed on the server, the TCP stack will send an RST to reject the establishment of the connection. According to TCP protocol requirements, when a SYN packet arrives at the server, its TCP stack will check if the target port is listening. If the port is unavailable, the connection request will become invalid. Therefore, based on this protocol specification, the project will use `nftables` to filter network packets before conducting experiments. This is a Linux kernel framework designed to provide flexible packet filtering functionality.

During the TCP three-way handshake, when the client sends a SYN packet, the server responds with a SYN-ACK packet upon receipt. When using Scapy to complete this step, it can be observed that the client automatically sends an RST packet upon receiving a SYN-ACK packet. This is because when using Scapy to send SYN packets, Scapy directly operates on the network interface, bypassing the TCP stack in the operating system kernel, so the TCP stack does not know that the sender is attempting to establish a TCP connection. After receiving a SYN-ACK packet from the receiver, the TCP stack does not manage such packets and instead responds directly to an RST packet. The reason why Scapy can bypass the TCP stack is because it uses raw sockets, which can access lower level packets and allow programs to write directly from user space to the network layer. In order to prevent the RST packet from affecting the experiment, the `Packet Filter` in MacOS, a firewall and traffic management tool, is used to filter RST sent to the server. Its implementation process is similar to the previous paragraph.

3.4 Default Parameters of TCP

In all tests, the client's Initial Sequence Number(ISN) was 724001 and the receiver's ISN was 17581102. The port used for communication by client applications is dynamically allocated and is commonly referred to as an ephemeral port. Its existence is temporary and only remains valid during the communication session. For the range of equatorial ports, 32768-60999 is used by many Linux kernels [12]. In this project, the range of temporary ports was randomly generated within the range of 49152-65535, as recommended by IANA [4]. In addition, when testing TCP packets, we set the Maximum Segment Size (MSS) of all packets to 512, which brings the following two benefits:

1. MSS specifies the maximum size of data segments that can be transmitted in TCP packets, excluding TCP and IP headers. In network paths with smaller MTU, when MSS is set to be small, fragmentation during transmission can be avoided, which increases processing overhead. Data packets that have undergone fragmentation need to be reassembled at the receiving end.

2. In some network environments with low bandwidth or high latency, using smaller MSS can reduce the burden on the network, improve the success rate of packet arrival, and reduce transmission latency.

3. Some old or limited performance network devices may not be able to efficiently process larger data packets. Setting MSS to 512 bytes can prevent these devices from experiencing issues due to receiving large data packets.

Chapter 4

Tests and Results

In all tests, we selected three different target port numbers for testing, namely 80, 443 and 49312. Port 80 is a standard port for HTTP, mainly used for unencrypted web page access; Port number 443 is a standard port for HTTPS, typically used to establish encrypted connections with servers, ensuring data privacy; Port number 49312 is a random port. Because the behavior of middleboxes may change depending on the destination port, testing two common ports and one uncommon port may reveal more behavior. For the tested network, we selected Wi-Fi from 20 public areas in the UK.

4.1 TCP Option Field

The options field in the TCP header is a variable length field located after a fixed 20 byte header, with a maximum length of 40 bytes. This field is used to support various extension functions and protocol optimizations, enabling TCP to improve performance and adapt to constantly changing network requirements.

Most of the widely deployed options currently start with the three-way handshake process, such as MSS, Window Scale, SACK, TCP Fast Open, and MPTCP. The client adds options in the SYN packet header to negotiate the use of options with the server. If the corresponding option can also be found in SYN-ACK, it indicates successful negotiation and the functionality will take effect in the session.

The behavior of middleboxes in the negotiation may have a negative impact, leading to the invalidation of options or inconsistent understanding of the negotiation results between the two ends. In order to reflect the impact of these behaviors in different scenarios as much as possible, the test is divided into three types:

1. Send a packet containing test options after a three-way handshake **(3WHS)**: In

this scenario, SYN and SYN-ACK do not contain any test options, and after a successful handshake, the client sends the first packet containing test options. The purpose of this test is to understand whether the intermediate box will allow new option fields to pass through during the data transmission phase when no new options are introduced during the handshake phase. Although this approach is very rare among widely deployed options, we would like to know to what extent middleboxes supports this approach.

2. The three-way handshake and data packet both contain testing options **(3WHS Plus)**: this test aims to evaluate whether the middle box will recognize and intervene in unknown option fields during the handshake phase, which helps to assess the level of support for middle boxes when new extensions are introduced. If the options in SYN are removed, this is acceptable, and SYN-ACK will not add new options, causing the TCP connection to revert back to the normal version. If the options in SYN are not removed and the options in SYN-ACK are removed, it will have an impact on both parties. At this point, the client considers the negotiation to have failed, and the service level will assume that the negotiation has been successful. In the last scenario, the new option can pass both SYN and SYN-ACK, but is removed in subsequent data packets. This situation will have an impact on both the client and server.

3. Directly sending packets containing test options **(Directly sending)**: In this scenario, we skipped the three-way handshake and sent packets containing new options directly. The purpose of this test is to observe how the intermediate box handles packets with new options without establishing a connection, especially whether they can pass through the network in a stateless state. This is crucial for evaluating scalability and compatibility in certain application scenarios, such as connectionless transmission.

In order to cover as many new extensions as possible in the future, we have divided the options used in the testing process into four categories:

1. **Unknown option**: We test it using option kind 35, which is marked as Reserved by IANA and belongs to the unallocated option.

2. **Experiment option**: We use option kind 253 for testing, which is marked as Experiment by IANA and is typically used to experiment with new features.

Table 4.1: 4 kinds of options in 3WHS

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	15	16	15
Removed	0	0	0
Changed	1	1	1

3. **Known option with wrong length**: The `Timestamp` option is used for testing, with an option kind of 8. Normally, the option length of TS is 10, but we change it to 16 during testing.

4. **Random value option**: We add a random value of 12 bytes to the option field and conduct testing.

In the options testing, we expect behaviors of middleboxes would include:

1. **Blocked**: In tests involving handshake, SYN packets may be intercepted, resulting that clients are unable to establish a connection. In the Directly sending test, data segments may be intercepted directly and the server cannot accept them.

2. **Passed**: All packets can pass through and the option field is not affected.

3. **Removed**: All packets pass, but the added options are removed.

4. **Changed**: Packets can pass, but options have been modified by middleboxes.

Tables 4.1 - 4.3 summarize the results including the handshake, where the situation where SYN is blocked remains the same. Among them, three paths will automatically respond with an RST with a seq of 0 upon detecting a SYN, and two networks will respectively block all SYN packets from ports 80 and 49312. It is worth noting that the middlebox that blocks all on port 80 is more special, because it will send a SYN-ACK for the client, but will not respond to the subsequent payload segments. Secondly, in tests that require establishing a handshake, a middlebox in a path will indiscriminately change the options in all segments. Detailedly, for SYN packets, they will be reset to MSS, SACK, TS, WScale options; And for the data segments, they will be reset to only include the TS option.

Table 4.2: Unknown, Experiment, TS in wrong length in 3WHS Plus

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	15	16	15
Removed	0	0	0
Changed	1	1	1

Table 4.3: Random value option in 3WHS Plus

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	13	14	13
Removed	0	0	0
Changed	3	3	3

The difference between Table 4.2 and Table 4.3 is that two networks recognized packets with random values option added. These two middleboxes will add a MSS option with a value of 536 at the beginning of the option field. And the originally added random values are not affected.

Table 4.4 summarizes the results in Directly sending. More than half of the networks directly block payload packets. The network that modifies data segments is the same as previously mentioned: all option fields will be reset to only contain TS options.

Table 4.4: 4 kinds of options in Directly sending

	Destination Port		
Behavior	80	443	49312
Blocked	11	11	11
Passed	8	8	8
Removed	0	0	0
Changed	1	1	1

Table 4.5: ISN modification test

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	15	16	15
Changed	1	1	1

4.2 Initial Sequence Number

Sequence number is the mechanism used by TCP for reliable transmission, packet ordering, and handling of duplicate data packets. The use of some TCP options may depend on the sequence number of the packet, such as SACK [17]. In the TCP three-way handshake phase, the SYN sent by the client and the SYN-ACK responded by the server each contain an ISN. If middleboxes modify ISN, it is likely to have an impact on this type of TCP option.

Table 4.5 shows the test results, where only one path has an impact on ISN in networks that allow SYN to pass through. In this path, middleboxes make bidirectional modifications to the ISN of both the client and server.

4.3 Sequence Number Hole

TCP's SEQ and ACK can be used to prevent packet loss in the network. Specifically, the receiver discovers lost packets by detecting gaps in the sequence number. For example, if the receiver expects the next sequence number to be 1001, but the sequence number of the received packet is 1201, then the receiver can request the sender to retransmit the lost packet through ACK. If the sender does not receive an ACK from the receiver within a certain period of time, or if the ACK received indicates that the receiver did not receive the data packet as expected, the sender will resend these data packets.

The purpose of the Sequence Number Hole test is to observe the behavior of middleboxes in this context. We divide the types of holes into two categories:

1. **Hole-in-SEQ**: First, establish a connection, and then the client sends two data segments in sequence. The `seq` of the two data segments contains a hole of size 500. Then observe the behavior of middleboxes to the second data segment.

2. **Hole-int-ACK**: First, establish a connection. The client sends the first data

segment, and then adds a hole of size 500 to the ACK value sent by the server. Observe whether the client can receive this ACK.

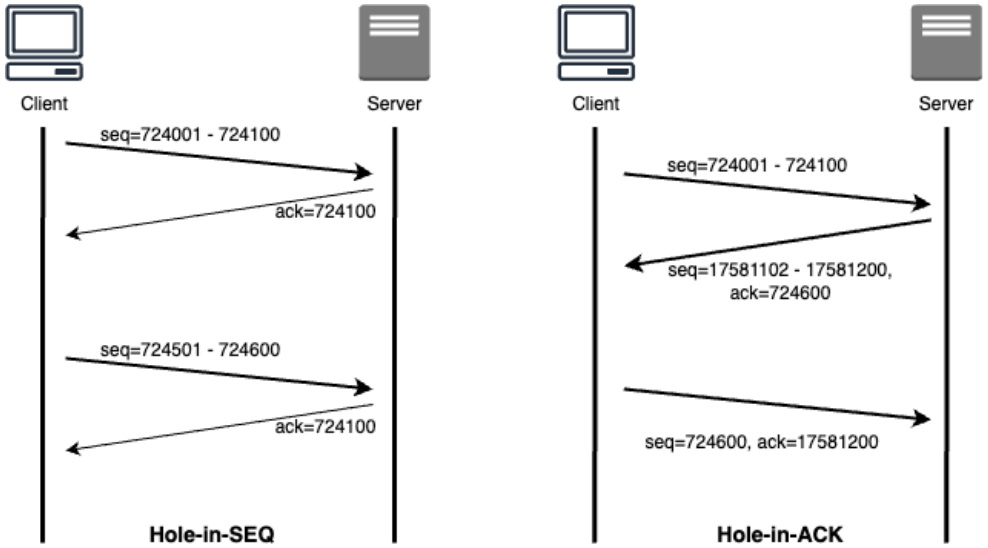

Figure 4.1: Sequence Number Hole Test

Figure 4.1 shows the differences between these two kinds of tests, and Table 4.6 - 4.7 shows the result. For Hole-in-SEQ, Fail represents that the client cannot receive an ACK for the second data segment. From the perspective of the server, the behavior of middleboxes causing failures can be divided into two types. The first case is that the server detects the second data segment with a hole, but the payload is removed by middleboxes and the corresponding ACK is blocked during the path; The second is that data segment with a hole are directly blocked by middleboxes, and the server cannot capture them.

In Hole-in-ACK, Fail represents that the client cannot receive the ACK with a hole. The behavior of middleboxes can also be divided into two types. Firstly, ACK with hole may be intercepted during transmission. And another behavior is more complex. When an ACK with a hole is sent from the server, the middleware will capture it and resend the previous data segment on behalf of the client. And the purpose of doing so can be speculated as hoping to receive packets with correct ack values. From this behavior, it can be seen that this is a stateful middlebox that stores recently packets.

Table 4.6: Hole-in-SEQ test

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	13	14	14
Fail	3	3	2

Table 4.7: Hole-in-ACK test

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	14	14	13
Fail	2	3	3

4.4 Retransmission

What choices can future protocol designers make if they find that the content of a previously lost packet has become invalid when they need to perform retransmission? Firstly, it can be seen from the previous test that sending packets with holes is not a very reasonable choice and may be affected by the network environment. Secondly, can the payload be modified before retransmission? To investigate the rationality of this choice, we conducted a retransmission test.

Figure 4.2 explains the process of the test. Firstly, the client sends two consecutive data segments with serial numbers, and then the server returns two ACKs confirming the first data segment. At this point, the client modifies the content of the second data segment and sends it. If the server can receive a complete and inconsistent data packet, it means the test has passed.

Table 4.8 shows the results, indicating that two networks failed, but the reason for the failure was not that inconsistent packets were blocked. In one of the tests, after capturing two duplicate ACKs, middleboxes merged the contents of the two data segments previously sent and sent them to the server. In addition, when middlebox receives the modified data packet, it will modify its contents. For example, the original payload length is `x` and the modified payload length is `y (y>x)`, the middlebox will cut off the first `x` and retain the content of the last `(y-x)`.

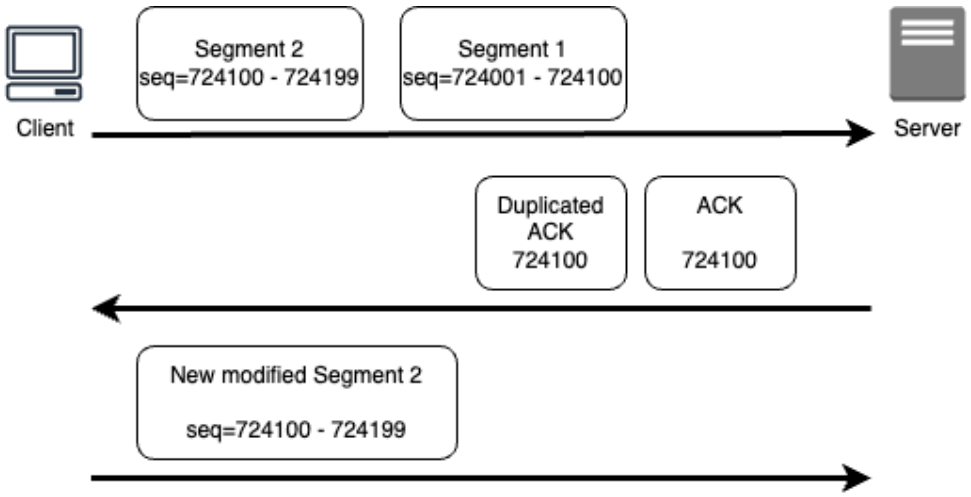

Figure 4.2: Retransmission Test

Table 4.8: Retransmission Test Results

	Destination Port		
Behavior	80	443	49312
SYN blocked	4	3	4
Passed	14	15	14
Fail	2	2	2

The behavior of another middlebox that causes failure is different. It does not block duplicate ACKs, but modifies inconsistent packets. Its operation is to concatenate the original content into the header of the modified content, and then send it to the server.

4.5 QUIC Tests

QUIC is a transport protocol built on UDP, serving as the foundation for HTTP/3. Unlike TCP's three-way handshake, QUIC's handshake is typically used to negotiate encryption parameters. According to RFC 9000 [11], we constructed an initial packet to simulate the handshake process of the protocol, and Table 4.9 shows the parameters used.

In tests, we observed potential behaviors of middleboxes towards the QUIC protocol in the current network environment by exchanging initial packets between the client and server. In the protocol specification, these two initial packets need to contain `ClientHello` and `ServerHello` respectively for negotiating TLS 1.3 encrypted ses-

Table 4.9: QUIC Initial Packet Parameters

Field	Length	Value
Header Form	1 bit	1
Long Packet Type	2 bits	0
Reserved bits	2 bits	0
Packet Number Length	2 bits	2
Version	1 byte	1
Destination Connection ID Length	1 byte	8
Source Connection ID Length	1 byte	5
Token	1 byte	0
Packet number	1 byte	random
Payload	259 bytes	random

Table 4.10: QUIC Initial Packet Test

	Destination Port		
Behavior	80	443	49312
Passed	13	13	13
Sending Packet Blocked	4	4	4
Response Packet Blocked	2	2	2
Changed	1	1	1

sions. However, during the package building process, we use random values instead of encrypted information.

Table 4.10 displays all the results. Unlike the results in the TCP section, the behaviors of all middleboxes remains consistent on different ports. There are four networks blocked initial packets from client. In TCP Option Field section, we mentioned that there is a network that blocks all TCP segments sent to port 49312, and it blocks all UDP packets in this test; And the other three networks blocked all TCP and UDP segments simultaneously. For the two 'Response Packet Blocked' results, the server can receive the initial packet, but the response cannot be seen by the client. There is a middlebox modified the payload, and in this case, the initial packet sent by the client remains unchanged. But after comparison, it was found that the content sent by the server was inconsistent with the response received by the client, its content was modified.

Chapter 5

Conclusion and Future Work

In this project, we evaluated the evolvability of transport protocols (TCP and QUIC) in the current Internet environment by active measurement. The test results indicate that although both TCP and QUIC were designed with expanding possibility, there are still many middleboxes in the real world that can affect the functionality of these protocols. These behaviors limit the updating and optimization of protocols. Specifically, our research found that:

1. The intervention of middleboxes on TCP options is modification. This type of device completely removes options from the data segment and adds its own set options, which can cause the new extension out of work.

2. More than half of the directly sent TCP segments are directly blocked, which is a huge challenge for those who want to try connectionless applications.

3. Some middleboxes may modify the ISN, which may affect TCP options that depend on sequence numbers.

4. When out-of-order packets appear, the behaviors of middleboxes may cause problems, which may result in packet loss or connection interruption.

5. Modifying the retransmission segments may not be a reasonable choice, as some middleboxes may modify inconsistent content.

6. Although the QUIC protocol reduces the intervention of intermediate devices through encryption, it still faces some challenges, such as middleboxes potentially blocking UDP traffic and modifying encrypted content.

Future protocol designers should carefully consider these potential issues when working, such as negotiating new features during the handshake phase, which is a good habit. If no new options are found in SYN-ACK, they should promptly push back to the regular version. At the same time, attention should also be paid to the risks of sequence number modification, segments with holes, and inconsistent retransmissions. In summary, extending the transport layer protocol is theoretically feasible, but it still faces challenges from the diversity of network environments.

The testing scope of this project is limited to 20 public areas Wi-Fi in the UK. These testing environments to some extent reflect the intervention behaviors of middleboxes in TCP and QUIC, but due to limitations in sample size and geographical scope, they cannot fully represent the situation in various network environments worldwide.

The future work should be to expand the measurement range and conduct testing on network services provided by different ISPs. By obtaining results in more diverse environments, it is possible to comprehensively measure the diversity of behaviors of middleboxes. Meanwhile, the global results can explain the regional differences in the behavior of different middleboxes, providing data support for future standardization expansion.

Bibliography

[1] D. Borman, B. Braden, V. Jacobson, and Richard Scheffenegger. Tcp extensions for high performance. *RFC*, 7323:1–49, 1992.

[2] Sapna Chaudhary, Prince Sachdeva, Abhijit Mondal, Sandip Chakraborty, and Mukulika Maity. Youtube over google's quic vs internet middleboxes: a tug of war between protocol sustainability and application qoe. *arXiv preprint arXiv:2203.11977*, 2022.

[3] Sarah Cook, Bertrand Mathieu, Patrick Truong, and Isabelle Hamchaoui. Quic: Better for what and for whom? In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.

[4] Michelle Cotton, Lars Eggert, Dr. Joseph D. Touch, Magnus Westerlund, and Stuart Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335, August 2011.

[5] Ryan Craven, Robert Beverly, and Mark Allman. A middlebox-cooperative tcp for a non end-to-end internet. *ACM SIGCOMM Computer Communication Review*, 44(4):151–162, 2014.

[6] Korian Edeline and Benoit Donnet. A first look at the prevalence and persistence of middleboxes in the wild. In *2017 29th International Teletraffic Congress (ITC 29)*, volume 1, pages 161–168. IEEE, 2017.

[7] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno and sack tcp. *ACM SIGCOMM Computer Communication Review*, 26(3):5–21, 1996.

[8] Benjamin Hesmans, Fabien Duchene, Christoph Paasch, Gregory Detal, and Olivier Bonaventure. Are tcp extensions middlebox-proof? In *Proceedings of the*

2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox '13, page 37–42, New York, NY, USA, 2013. Association for Computing Machinery.

[9] Benjamin Hesmans, Fabien Duchene, Christoph Paasch, Gregory Detal, and Olivier Bonaventure. Are tcp extensions middlebox-proof? In *Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization*, pages 37–42, 2013.

[10] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 181–194, 2011.

[11] Jana Iyengar and Martin Thomson. Rfc 9000: Quic: A udp-based multiplexed and secure transport. *Omtermet Emgomeeromg Task Force*, 2021.

[12] kernel.org. Ip sysctl. https://www.kernel.org/doc/html/latest//networking/ip-sysctl.htmlip-variables. Accessed: 2024-08-11.

[13] Mike Kosek, Tanya Shreedhar, and Vaibhav Bajpai. Beyond quic v1: A first look at recent transport layer ietf standardization efforts. *IEEE Communications Magazine*, 59(4):24–29, 2021.

[14] Mike Kosek, Benedikt Spies, and Jörg Ott. Secure middlebox-assisted quic. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2023.

[15] Mirja Kühlewind, Sebastian Neuner, and Brian Trammell. On the state of ecn and tcp options on the internet. In *International conference on passive and active network measurement*, pages 135–144. Springer, 2013.

[16] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[17] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. Tcp selective acknowledgment options. RFC 2018, 1996.

[18] Alberto Medina, Mark Allman, and Sally Floyd. Measuring interactions between transport protocols and middleboxes. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 336–341, 2004.

[19] Késsia Nepomuceno, Igor Nogueira de Oliveira, Rafael Roque Aschoff, Daniel Bezerra, Maria Silvia Ito, Wesley Melo, Djamel Sadok, and Géza Szabó. Quic and tcp: A performance evaluation. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00045–00051, 2018.

[20] ostinato.org. Ostinato traffic generator for network engineers. https://ostinato.org/. Accessed: 2024-08-11.

[21] Pellegre. libcrafter. https://github.com/pellegre/libcrafter. Accessed: 2024-08-11.

[22] Michele Polese, Federico Chiariotti, Elia Bonetto, Filippo Rigotto, Andrea Zanella, and Michele Zorzi. A survey on recent advances in transport layer protocols. *IEEE Communications Surveys amp; Tutorials*, 21(4):3584–3608, 2019.

[23] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. Tcp fast open. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, pages 1–12, 2011.

[24] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A first look at quic in the wild. In *Passive and Active Measurement: 19th International Conference, PAM 2018, Berlin, Germany, March 26–27, 2018, Proceedings 19*, pages 255–268. Springer, 2018.

[25] Salvatore Sanfilippo. hping. https://github.com/antirez/hping. Accessed: 2024-08-11.

[26] Scapy.net. Scapy. https://scapy.net/. Accessed: 2024-08-11.

[27] Michael Walfish, Jeremy Stribling, Maxwell N Krohn, Hari Balakrishnan, Robert Tappan Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *OSDI*, volume 4, pages 15–15, 2004.

[28] Wireshark Foundation. Wireshark – go deep. https://www.wireshark.org/, 2024. Accessed: 2024-08-11.