Measuring transport layer evolvability in the Internet

Shashwat Pushparaj Mujumdar

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

Internet's transport layer is responsible for data transfer in the form of packets across communication networks. Intermediary components, known as the middleboxes are claimed to be responsible for influencing the behaviour of many transport layer protocols and influence packet transmission to an extent. Therefore this project is focused on evaluating scalability of Internet's transport layer, through the generation of weird packets using Scapy. Initially, packets with customized header were crafted and a client-server infrastructure was built to simulate the usage of novel header transport protocol, followed by packet capture on server end, to check for any modification whilst network traversal. Core components of the project deal with implementing a non UDP/TCP header transport protocol for data transfer, along with checking its validity across distinct network types as well as exploring TCP behaviour by restyling its options. Final emphasis was on extending UDP packets to replicate QUIC protocol, changing its fields to investigate possibility of a QUIC replaceable protocol's existence and perform tests by varying its header parameters to observe outcome of middlebox interaction across heterogeneous networking environments.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

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.

(*Shashwat Pushparaj Mujumdar*)

Acknowledgements

I want to sincerely express gratitude towards my project supervisor, Prof Michio Honda for guiding me throughout the course of dissertation and provide valuable feedback. His motivation and thorough knowledge played an important role in attaining the intended goals of the research project. I would also like to acknowledge, the contribution of University of Edinburgh in providing me with the opportunity to engage in this profound research work, which has helped me to procure novel technological know how.

Lastly, I am also grateful towards my family and friends, who have encouraged me continuously. With their confidence in my abilities and constant moral support, I was able to complete my tasks successfully and remained determined throughout this interesting journey.

Table of Contents

Chapter 1

Introduction

Traditionally, User Datagram Protocol and Transport Control Protocol have been the central transport layer communicating protocols which are widely used for data packet transmissions across different networks. With growing usage of modern technologies, complexity of networking environments has increased, especially due to introduction of multiple intermediary entities in the communication channel termed as middleboxes.Transport layer is heavily influenced by these middleboxes, resulting in its efficiency being compromised. Middleboxes like firewalls, NATs can cause packets to get dropped or changed en-route thus hindering functionality of transport layer. To get deeper insights about middlebox and packet interaction, several networks were tested using client-server architecture so that transport layer protocols can be scaled up for better operational performance.

1.1 Background and Motivation

Integration of 21st century communication technologies such as 5G and virtual storage facilities like clouds have created necessity of vigorous information sharing protocols. End-to-end transmission of packets has increasingly become complicated ever since. Based on the studies by [13][16][19] it is evident that evolvability of the transport layer protocols has some serious limitations. Robustness of network is affected by the presence of middleboxes within the communication channel between interacting machines. Evolvability of transport layer protocol, faces limitations such as ossification when intermediate components within a network influence data units.

Explicit Congestion Notification is a characteristic of TCP extensions which is another developmental constraint within connection frameworks. Discovery of novel protocols as well as modification of currently existing transport layer protocols is being carried out to meet the rapidly growing demand of modern users. Therefore it becomes critical to understand how novel header packets are treated by middlebox entities, which were crafted during the project work using the packet generation and manipulation tool supported in Python i.e. Scapy.

1.2 Project Objectives

The fundamental objective of the project work was to evaluate how scalable the transport layer of Internet is, when non-standard packets are transmitted.Secondarily, the goal was to build a client side framework and send weird-packets that are crafted manually to a virtual machine. On the virtual machine kernel using tcpdump the packet header information was to be analysed, so that packet transmission could be verified. Following this, the server side framework would be designed to receive and capture packets, so that any changes in their header due to network standardisation could be detected. Validity of a transport protocol was to be assessed across distinct networks.

Exploration of TCP behaviour, on changing its options, was another goal which was delegated to my project colleague. Final objective of project was to also experiment with UDP packets that were mimicking QUIC protocol behaviour, alter its header parameters and conclude whether a replacement to current QUIC protocol was possible, after inspecting impact of network intermediaries on these packets, again across large number of interactive systems.

1.3 Project Scope

The research is concentrated on implementation and development of non-traditional transport layer protocols and evaluate how the packets are modified when they reach the receiver. Based on the packets received a response packet would be sent back to the sender machine by the receiver. Throughout this process, how the initial packets are modified during network traversal, and how the response packets compare to those received by the receiver was the main area of interest.

Across different wireless networks, multiple variations in the header structure were carried out such as changing the protocol number, varying the transport payload fields etc. While the unique QUIC protocol, that resembled the traditional QUIC protocol had its fields messed up, the middlebox effect on it was to be evaluated when we send packets across the communication channel.

1.4 Project Results

The project would reveal about the custom packet alterations that were caused by middleboxes,such as changes in data payload, fields of header and ports/IP addresses. Tests across multiple networks would display the compatibility and feasibility of the protocol in real world deployment. Adjustment of TCP options will provide in-depth picture of how TCP's performance would change when affected by intermediate entities within the network. Extending the study further, UDP packets would be designed in such a way that they replicate QUIC protocol.

Final outcomes of these packets when being tested across different networking environments would display constraints of how well it would replace QUIC in practical use cases. Overall the adaptability of novel transport protocols to middlebox influence and their robustness, would be judged so that effective protocol mechanisms can be adopted across diverse networks.

1.5 Dissertation Organization

The dissertation is divided into five chapters,exploring through the previous work, the implementation methodology and technical details, analysis of the results obtained and lastly conclusions drawn about the scalability of the Internet's transport layer. The first section , i.e the Introduction lays the foundational significance of transport protocols in packet transfer and effect of middleboxes. Further it expands on research scope, objectives and the expected results after testing. The second chapter will revolve around the current research work and its relation to the project's objectives.

Third chapter highlights on the project methodology details and actual implementation. Packets are generated by Scapy, describing customized header packets which are different from traditional TCP and UDP. Client-server infrastructure would be used to simulate packet reception and response to understand whether packets are dropped modified. Over the current UDP packet fields, unique alterations would be carried out to make it similar to original QUIC protocol and then mess with its fields to get interesting patterns about packet-middlebox interplay.

Fourth chapter of the report would be dealing with research findings, also diving in the analytical aspect. Non-standard packets would be gauged for whether their headers are changed by middleboxes, whether are blocked or reach intact between client and server. During testing fields would be varied for the customized packet. In the fifth chapter, critical conclusions would be represented post testing and throw light on how future research work could be inferred. Collectively this section will not only provide a brief about key findings of project but also call attention to contribution made in the field of transport layer evolution.

The fifth chapter is also based on Future research , which will showcase how this piece of work can influence development of new protocol structures as well as suggest innovative approaches to handle extensions of standard transport protocols like TCP.

Chapter 2

Previous Work

2.1 Transport layer scalability challenges

Transport layer's flexibility has been a topic of great debate among the rapidly growing user demand and emergence of new applications [12]. The works by [19] and [20] describe transport layer protocol behaviour, multi-network ecosystem flexibility and the limitations that are faced in deploying them to enhance communication. Firewalls are often responsible for blocking the unnecessary data packets in a communication link to manage the data traffic.

Network proxies also can cause congestion, thus degrading quality of data. Moreover, the research works also present how TCP responds to changes in the kinetics of diverse networks. A tussle is created by the middleboxes, as the deployment of novel transport protocols is not much supported by middlebox dealers until those protocols don't gain a widespread acceptance among the users. This lack of motivation has flattened the rate at which modified protocols would be adopted, resulting in traditional protocols like TCP and UDP being major players.

Some innovations like enhancement of transport API, use of happy eyeball frameworks for end to end determination of protocol validity have been proposed in [23] to minimize ossification. [13] brought into limelight the necessity to employ important insights so that extensibility of TCP can be improved to overcome challenges. With these insights, the transport layer protocols can be redesigned to perform better and establish feature control accurately. [9] explored constraints of using IP options across different networks. Middleboxes have the tendency to often mess with IP options, many times dropping the packets in which they are embedded straightaway. Middlebox architectures are designed to promote security and strict compatibility compliance for the protocols, resulting in hindered IP option utility. Our project extends this research further by incorporating disrupted header fields into the test scenarios for standard transport protocols such as TCP,UDP and QUIC, underscoring the necessity for betterment of protocol design, as majority of network features are influenced by intermediate entities within the communication channel.

Existence of various issues in network such as interference, congestion may cause ossification of the transport layer which is a serious concern. NEAT is an example of a modern architecture designed for networks, which is specifically aimed to separate out applications and uplift services offered by transport layer to initiate constructive modification in interactive environments[12].

2.2 Analysis of TCP and its extensions

The significance of congestion control in networks and obstacles in packet traversal was noted by [16], besides explaining methodologies to eliminate these connection issues. [13] has presented important information regarding TCP headers and options for evaluating the transport layer evolvability. A thorough study of 142 networks was carried out, whereby the networks were actively evaluated to witness how middleboxes influenced TCP extensions. Although the study concludes that TCP can be extended using options, they should take into account currently persisting behaviours of network middleboxes. Within the study it was observed that sequence numbers of commuting packets maybe changed or only partially allowed, others being blocked from the TCP protocol's movement.

This made it notable to use relative sequence numbers instead of non-variable ones. Segmentation of TCP and large receive offload were also surveyed for determining the way they influenced TCP option handling process. Based on the survey it could be pointed out that options should be manipulated in such a way that they can bear duplication and elimination during segment splitting and coalescing procedures.Furthermore design specific alternatives consisting of Multipath TCP and TcpCrypt have been navigated through to provide assistance in managing middlebox inflicted restrictions.

2.3 Analysis of UDP and its role in designing novel protocols

A study [8] comparing UDP performance in contrast to TCP, and UDP's potential to be used as a foundation to develop novel protocols revealed that around 5 percent of total terrestrial networks can absolutely block UDP packets. Moreover, the research also claims that UDP packets having larger length are less likely to be blocked while travelling through any network. UDP impairments generally occur in real-world enterprise environments, or else geographical regions having weaker network connectivity. The need for UDP traffic to navigate through NAT is essential, to avoid timeouts from affecting protocol usage in wireless communication systems. UDP can be used for building encapsulated transport layer protocols, by implementing racing methods, to promote better adjustment with network fluctuations. The study claimed that UDP is a feasible option to be used as building block for new transport protocols like QUIC, but needs fallback methodologies in case of poor affinity.

2.4 Middlebox-Protocol interaction measurement

Based on the research by [19], the impact of middleboxes like NATs, firewalls on new transport protocols and their features was examined. Middleboxes interfere with the communicating packets and violate the Internet's end to end principle [25], which states that a low-level function's importance in a system is insignificant when compared to the cost required to use it at that level. TBIT tool was used for testing distinct web servers and determine their reaction to multiple TCP extensions. Unexpected behaviors could be analyzed when transport layers interacted with middleboxes. TCP extensions like Selective Acknowledgement, Timestamps and ECN negotiation were utilised, following which it was demonstrated that middleboxes do cause unexpected changed in correct behaviors of TCP framework, mainly because of non-passive filtering processes or else due to the modifications of TCP mechanisms.

Subsequently ,in the research it was also argued that ossification caused by middleboxes have led to ambiguity in deployment of novel protocols and also in extending those that are currently accepted. Finally, some approaches to reduce this ossification have been put forward by the researchers, comprising of middlebox-proof transports.

2.5 Analysis of QUIC protocol

QUIC is one of the most prominent novel protocols besides others like SCTP and DCCP, known for having brilliant compatibility to most recent developments in technology in correspondence to the ever increasing requirements of users. QUIC protocol has been specifically designed to address the need to improve traditional TCP protocol's efficiency and minimize latency. It uses single round trip to reduce connection time and multiplexes streams.

Additionally error detection mechanisms along with recovery, offer highly reliable data transmission as compared to UDP. End to end encryption is supported by QUIC , which significantly enhances security while transmitting data. Overall QUIC has better adaptability to rapidly changing network environments and thus is a promising alternative to UDP[15].

The executable efficacy of TCP and QUIC was investigated over websites and webpages by [22], based on which it was concluded that TCP was more faster, thus reflecting QUIC protocol's unsatisfactory performance as well as show the need to optimize it.

On the contrary, according to the research work of [6], when QUIC and TCP were tested along with HTTPS and UDP protocols for executing the task of webpage retrieval, QUIC indicated to have better throughput. Consequently QUIC revealed the constraints of networks that were deformed. QUIC's higher robustness in comparison to TCP whilst handling packet loss or drop within the communication network was highlighted in the analysis carried out by [17]. On the application level, QUIC can be used to refine metrics, which could be extremely beneficial in real-world networking ecosystem.[2] evaluated QUIC under controlled environment as well as real-world environment, including network problems like loss of packets and bandwidth limitations. Based on this research we were able to explore how robust QUIC's header would be when prone to middleboxes, when compared with TCP or UDP. QUIC's multiplexing capabilities, can prevent blocking at head-of-line range, therefore demonstrating diverse variations on interaction with middleboxes. The possibility of finding a QUIC replacement protocol would be examined, along with its design and resilience under different network environments being evaluated against the traditional QUIC behaviour.

2.6 Test tools for transport protocols

Scapy, is the Python programming language's library which is used for crafting weird packets and manipulating packet structures [7]. It was used by [24] to test across many security applications, thus showcasing its high resilience in conjunction to exploring the packet manipulation abilities while testing the client-server framework. Scapy supports conversation friendly language,also enabling multitasking, use of high level functions and modularity.

Another perk of using scapy is that it validates a high range of different protocols such as TCP , UDP, ARP and ICMP. Novel protocols can also be defined as Scapy has the features for inheritance of layers using classes. Moreover it also allows for new fields to be defined while designing a new protocol. Scapy has three fundamental functions,comprising of packet construction and transmission, sniffing and incoming response packet. Wireshark can be linked to Scapy for analysing packets.

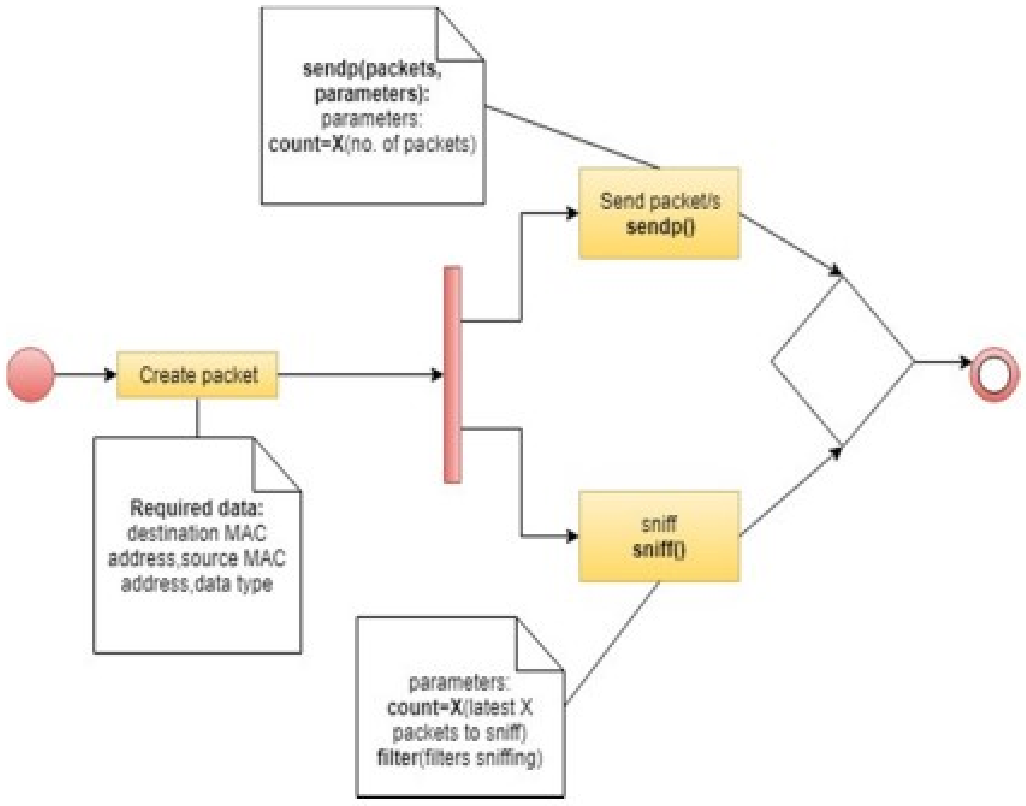


Figure 2.1: Working of Scapy [24]

A comparative study between Wireshark and tcpdump was conducted [11] , based on which some key features of these tools, which are simple to be used for network analysis were identified. Tcpdump, enables users to dump raw packets, which can give a clear overview of TCP packet fields. Wireshark is widely renowned, for its graphical interface, which makes it easy to understand. In addition to the interface, Wireshark also has strong packet capturing and filtering choices, thus helping it scan through distinct types of networks like Ethernet and Wifis.

Tcpdump as compared to Wireshark has less memory constraints, its virtual size being between 27-29 MB, whereas Wireshark has virtual memory ranging from 620 MB to 640 MB. However, Tcpdump is more efficient than Wireshark when power usage is considered. Tcpdump has battery usage of almost 0 percent, and Wireshark's consumption is around 0.2 percent. Number of wakeups in tcpdump is lower than the number of graphics wakeups in Wireshark. Wireshark is faster as compared to tcpdump,when capturing packet over high traffic networks. If the capture rate in network is very high usually tcpdump drops higher number of packets than Wireshark does. Tcpdump can hence be optimized using snaplen to avoid large quantity of packet from collapsing.

User friendly interface is a plus point held by Wireshark, coupled with colour encoded packet separations in contrast to raw Tcpdump output. For rigorous and in-depth analysis Wireshark is vital among users, but to gain a better fundamental background knowledge about networking tcpdump is a better option.

Chapter 3

Methodology and Protocol Testing

3.1 Transport layer protocols

3.1.1 TCP and UDP headers

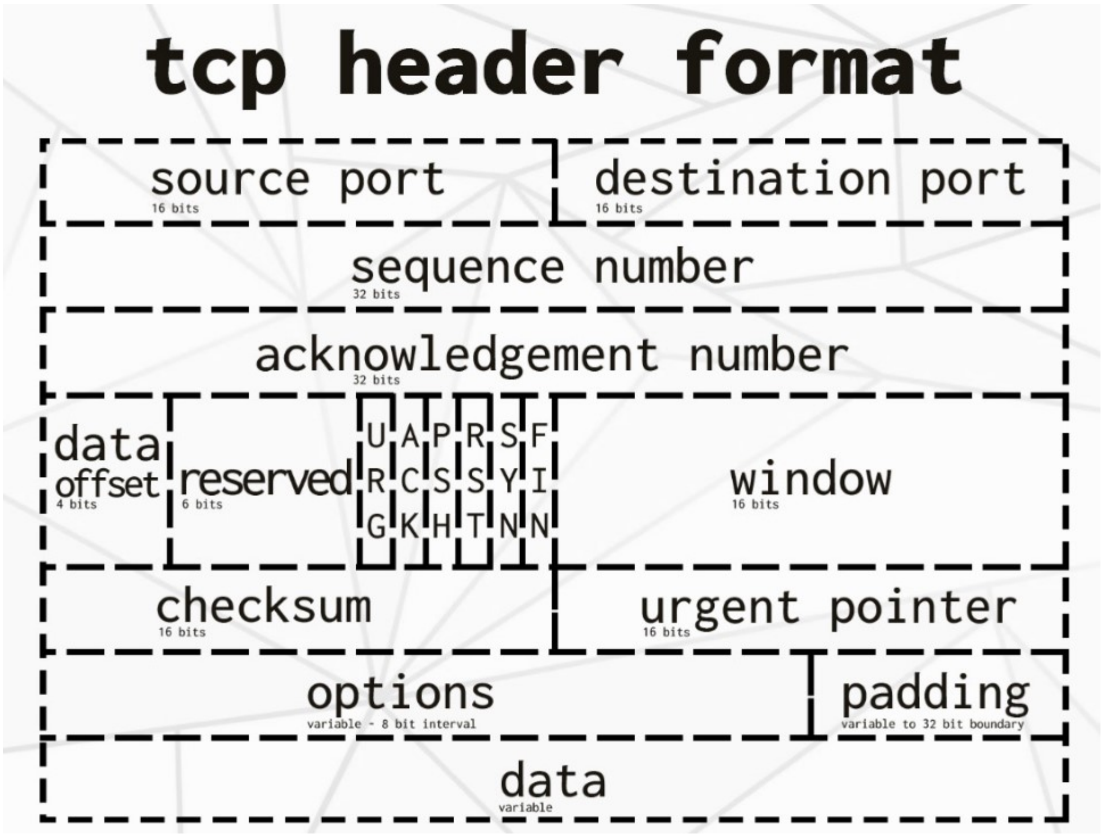


Figure 3.1: TCP header [3]

TCP is an important protocol, which has predominantly facilitated communication among different systems on any network. Its header length ranges from 20 to 60 bytes, including fields like source and destination port addresses, which play a crucial role in accurate data transfer. Sequence number, guarantees that data packets are transmitted in right order, whereas Acknowledgement number sends a confirmation of data being received.

Data offset field points to the total length of TCP header. The Flag field mainly deals with data transmission, connection establishment and terminating the connection. Window size determines the amount of data that the receiver can hold. Error handling is provided by the Checksum field, ensuring data integrity, and the data prioritisation is handled by the urgent pointer field, when multiple data points are being tackled in a segment.

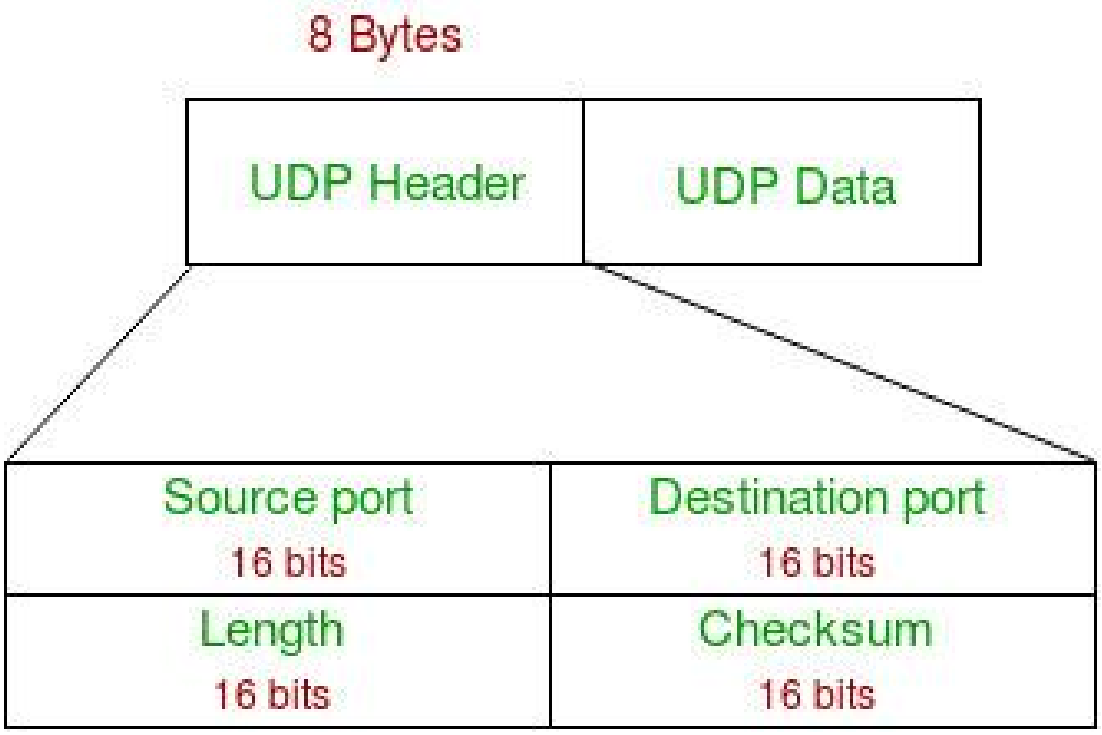


Figure 3.2: UDP header [10]

UDP header has a simple structure, with 8 bytes, mostly used for wireless communication. The four major fields of the UDP header are source port, destination port, length and checksum. Similar to TCP, source and destination ports play a crucial role in correct data transfer. Checksum field again plays the same role of error handling and detecting modification of data if any occurs during transmission. Overall length of the UDP packet header, along with the data is specified by the Length field.

Due to absence of sequence and acknowledgment fields in UDP, it has higher agility compared to TCP. But due to no check on packet sequence, it lacks reliability, as a result of which packet reception is not assured.

3.1.2 QUIC protocol header

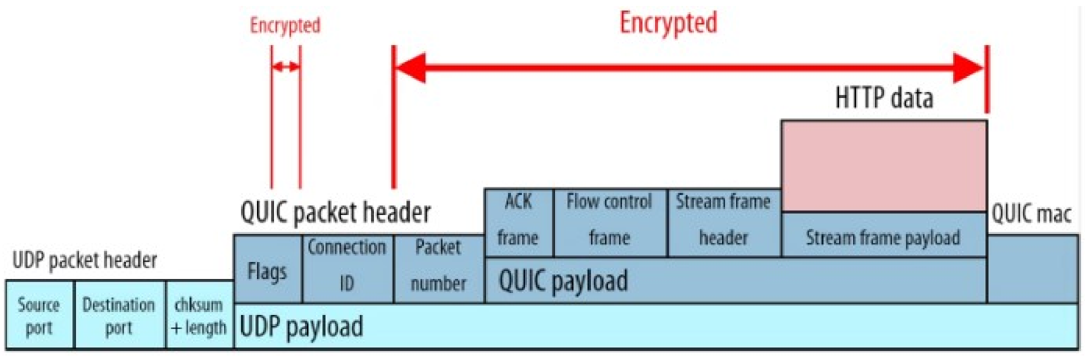


Figure 3.3: QUIC header [18]

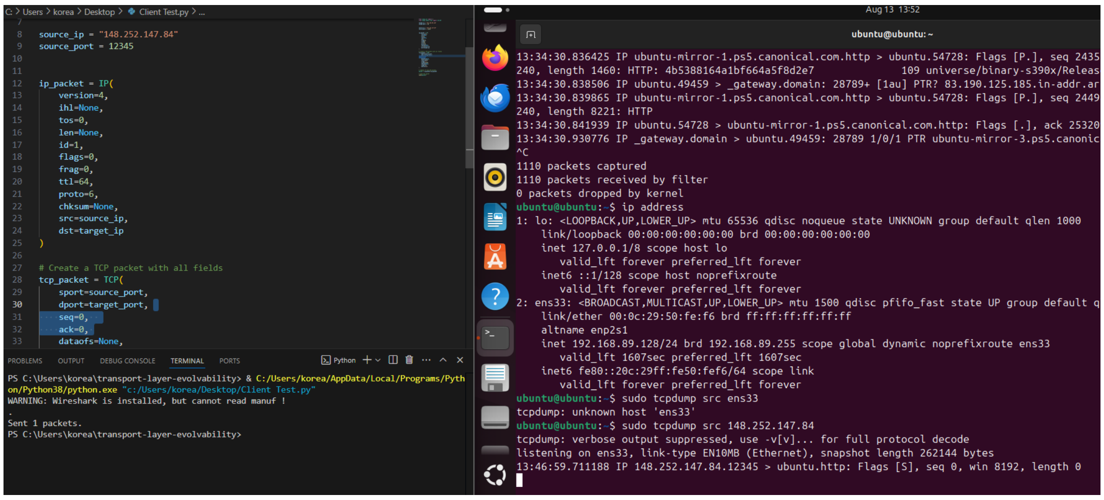
QUIC's header has a UDP packet header as a prefix and itself is embedded in UDP payload. QUIC's header includes the Connection ID field, which is used for managing establishment of communication links across diverse networks. The Packet Number field, guarantees that packet order remains intact, by assigning a unique number to each of them for authentication purposes, thus enhancing data security.

Version and Flags field maintain compatibility among several editions of QUIC and highlight how connection will be setup by the packet respectively. QUIC payload supports multiple types of frames such as ACK and STREAM, which play a major role in the data encryption process.

3.2 Testing ecosystem

3.2.1 Client-side implementation

Initially, raw packets were crafted using Scapy tool, having a IPV4 header followed by TCP header. On the client side firstly traditional TCP header packets were designed, and sent to a virtual machine. The Virtual machine used was VMWare with a Linux interface. tcpdump tool was run on the VM terminal to sniff and capture packets. Header details of each packet received were carefully analysed for detecting any discrepancy in the initial header fields of the packets.



```python
source_ip = "148.252.147.84"
source_port = 12345


ip_packet = IP(
    version=4,
    ihl=None,
    tos=0,
    len=None,
    id=1,
    flags=0,
    frag=0,
    ttl=64,
    proto=6,
    chksum=None,
    src=source_ip,
    dst=target_ip
)

# Create a TCP packet with all fields
tcp_packet = TCP(
    sport=source_port,
    dport=target_port,
    seq=0,
    ack=0,
    dataofs=None,
```

```
PS C:\Users\korea\transport-layer-evolvability> & C:/Users/korea/AppData/Local/Programs/Pyth
on/Python38/python.exe "c:/Users/korea/Desktop/Client Test.py"
WARNING: Wireshark is installed, but cannot read manuf !
.
Sent 1 packets.
PS C:\Users\korea\transport-layer-evolvability>
```

```
13:34:30.836425 IP ubuntu-mirror-1.ps5.canonical.com.http > ubuntu.54728: Flags [P.], seq 2435
240, length 1460: HTTP: 4b5388164a1bf664a5f8d2e7           109 universe/binary-s390x/Releas
13:34:30.838506 IP ubuntu.49459 > _gateway.domain: 28789+ [1au] PTR? 83.190.125.185.in-addr.ar
13:34:30.839865 IP ubuntu-mirror-1.ps5.canonical.com.http > ubuntu.54728: Flags [P.], seq 2449
240, length 8221: HTTP
13:34:30.841939 IP ubuntu.54728 > ubuntu-mirror-1.ps5.canonical.com.http: Flags [.], ack 25320
13:34:30.930776 IP _gateway.domain > ubuntu.49459: 28789 1/0/1 PTR ubuntu-mirror-3.ps5.canonic
^C
1110 packets captured
1110 packets received by filter
0 packets dropped by kernel
ubuntu@ubuntu:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default q
    link/ether 00:0c:29:50:fe:f6 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.89.128/24 brd 192.168.89.255 scope global dynamic noprefixroute ens33
       valid_lft 1607sec preferred_lft 1607sec
    inet6 fe80::20c:29ff:fe50:fef6/64 scope link
       valid_lft forever preferred_lft forever
ubuntu@ubuntu:~$ sudo tcpdump src ens33
tcpdump: unknown host 'ens33'
ubuntu@ubuntu:~$ sudo tcpdump src 148.252.147.84
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
13:46:59.711188 IP 148.252.147.84.12345 > ubuntu.http: Flags [S], seq 0, win 8192, length 0
```

Figure 3.4: Client-side testing for traditional TCP packets using tcpdump on VMWare

In the next stage, the client side was built using a baremetal linux on Cloudlab which is a virtual communication testing workplace. Using scapy, again the TCP header of the packet was varied with a novel protocol number, and broken header fields. Cloudlab was an efficient mechanism to test packet transfer in a controlled environment.

Due to changed protocol number, the weird packets crafted were partially received by the tcpdump run on the receiver baremetal system on Cloudlab across multiple networks due to unidentified protocol type. Furthermore, the packets were crafted in a real-world environment on a Windows OS machine, and the server side was kept on the Cloudlab.

As an outcome of real-world environment packet transfer, no packets were received on the Cloudlab server. Wireshark was used to filter packets based on the interface from where the packets were being crafted and the source IP address. However due to the completely null reception of modified TCP header packets, with novel protocol numbers like 253 and 254, used especially for experimental purposes, TCP packets having protocol number 6 [4] which is the same as traditional TCP protocol was used.

Additionally, client side implementation was refined with novel variations in TCP header fields, instead of varying protocol number in IP header.User Datagram protocol packets with changed header fields were also crafted to understand how they interact with network middleboxes. Lastly QUIC protocol replacement was tested, by writing an application over UDP socket, therefore eliminating the need to craft raw packets in this case.

3.2.2 Server-side implementation

Server side implementation was first simulated over the Cloudlab platform, to understand how efficiently it captured the packets. Even though packet sniffing was the fundamental objective, sending back response packets to the client machine was also necessary. Using scapy, the weird packets crafted by the client on Cloudlab, were captured on the server machine.

The server machine was enhanced with a filter to only consider packets holding the source IP. When the packets were received by the server machine, it sent response packets back to the sender side, thus showcasing packet reception and receiver acknowledgment.

Secondarily, when testing in real-world environment, the receiver was failing to receive any TCP packet , with changed protocol number. But with the new refinement in TCP header, which kept the protocol number as 6, server-side received packets and sent back response packets to the client across multiple networks. Following successful implementation for TCP header packets with broken header configurations, receiver side was also built for UDP and QUIC replacement protocols. For the case of UDP, no response packets were crafted, as UDP doesn't support three way handshake like TCP [21].

3.3 Testing Challenges

In this project, primary task was to test TCP protocol, with certain variations in its fields, to simulate novel protocol behaviour and differentiate it from traditional TCP packet interaction across different networks. To inculcate new protocol characteristics within the TCP header, three cases were considered, which would significantly disrupt the regular TCP packet's behaviour. Initially the emphasis was to change the protocol number in the IP header of the crafted packet, which would indicate novel protocol.

Additionally in the first case the IP header length and the data offset were to kept consistent with each other. Followed by that, the first 32 bytes of the TCP header were kept intact, i.e the source and destination port addresses, with all other fields completely being randomized.

In the final case all the TCP header fields were kept random, accompanied with a new protocol number. But due to the network being unable to identify the protocol type, all the packets with protocol number 253 and 254 were being dropped.

3.4 Novel protocol design

To enable, packet transfer a new refinement was made to the packet's header structure. Based on this refinement again three cases originated, which were to be tested for checking how the packets were treated by middleboxes. Whilst testing the TCP header, sequence number were broken, data offset was broken and lastly the source port address was randomized. Each of the case was tested over three different destination ports, which were 80, 443 and 50000.

This was in accordance with the research by [13], where the port 80 and 443 were chosen, which support HTTPS and HTTPS traffic respectively. 50000 port number is not assigned to any service by the IANA and thus was chosen as the third destination port for understanding how an unassigned port would behave on packet reception.

Similarly, for UDP protocol tests, the header structure was modified to simulate novel protocol behaviour. Two cases were considered, first being a broken header length field of the UDP packet and second being random source port numbers. Both of these cases were again tested across three different ports, 80, 443 and 50000. The QUIC protocol is built on a UDP application, hence its header is encapsulated within UDP payload. The fields for UDP payload were completely broken by randomizing all the values being assigned.

QUIC was tested across three destination ports 80, 443 and 49312. There were three potential results that would have been observed, one was packet header being modified by the middleboxes, second packets being blocked totally, and third packets being partially transmitted or received. According to the study conducted by [1], middleboxes can have both negative as well as positive impact on data flow. Connections using TCP transport can experience reliability and state persistence problems, if firewalls or other interruptions are present.

3.5 Initial Parameters and experimental procedure

3.5.1 Procedure

Broken source ports, broken sequence numbers and broken data offset were tested across 20 networks for TCP protocol. The source IP address, destination IP address and protocol number were passed as command line arguments. For each destination port the header structure of packets sent by the client machine, packets received by the server side and the response packets sent back by the server were recorded, and analysed for change in their default header field values. 9 networks were tested across Glasgow, in addition to 11 other networks across Edinburgh.

Networks having distinct ISP were preferred while testing as it would promote a wider range of possible results and better analysis of middlebox-TCP interaction. Broken length and source ports for UDP were also tested across 20 networks.

The packet header fields were observed at the sender side as well as the receiver side, to record any change in header fields while the UDP packet had traversed through the network. Finally, QUIC protocol's UDP payload fields were randomized to understand how it was affected by middleboxes.

TCP behaviour was tested across multiple ports, as with different destination ports it is variable in nature [13]. For each test, the changes in TCP, UDP and QUIC header would be noted at both the sender and receiver side. For TCP tests, the sender code would be sending packets with SYN flag as displayed in [13] and the response packets sent back by the receiver side on reception of packets would have SA flag for sending back acknowledgement.

3.5.2 Initial Parameters

1. **TCP broken sequence number experiment**

IP Header Fields	**Value**
version	4
ihl	5
tos	0
id	54321
frag	0
ttl	64

Table 3.1: IP Header Fields

TCP Header Fields	**Value**
sport	Command line variable
dport	80,443,50000
flags	S (SYN)
seq	1000 + (i * 1000) + random_offset
ack	0
dataofs	5
reserved	0
window	8192
urgptr	0

Table 3.2: TCP Header Fields

This experiment was focused on simulating TCP packet behaviour, specifically with an emphasis on how randomization of sequence number would have effect. Source port, source IP, destination IP address and protocol number were all command line arguments. Protocol number was maintained 6, and all other user defined arguments were variable across 20 different networks. Sequence number(seq), was allocated a formula, which generated values starting with 1000, changing further with random offset value ranging between 1 to 5000 and an index that is iterating.

Checksum for both headers are set to default None. Flags are set to S, i.e SYN flag for initiating connection. No acknowledgement, standard length of data offset and null value for reserved bits can be noted from the default values. Window size of 8192, is an appropriate window size for proper flow management.

2. **TCP broken data offset experiment**

IP Header Fields	Value
version	4
ihl	5
tos	0
id	54321
frag	0
ttl	64

Table 3.3: IP Header Fields

TCP Header Fields	Value
sport	Command line variable (src_port)
dport	80,443,50000
flags	S (SYN)
seq	1000
ack	0
dataofs	0 (introduces broken dataoffset)
reserved	0
window	8192
urgptr	0

Table 3.4: TCP Header Fields

The second case for simulating novel protocol behaviour over TCP packets was to set the data offset to 0 value. As data offset is usually a pointer to the entire header length of a TCP packet, representing it in 32 bits. Its valid value, which is minimum 5, indicating 20 bytes size when changed to 0, may cause data within the packet to be misinterpreted or even result in packet being dropped. IP header field values are kept same as in previous case, and a sequence number of 1000 is assigned to ensure correct ordering of data packets during traversal.

Source port is again a user-defined value, as also are the destination and source IP addresses. Checksums are set to None by default for both the headers, besides destination ports being varied over values 80,443,50000.

3. **TCP broken source port address experiment**

In this case, source ports are assigned in a random manner, dynamically selecting any value between 0 to 65535. With this range, not just ports that are reserved, but also experimental ports are tested while analyzing how broken ports effect packet transmission. Real-world scenarios, extensively use ephemeral ports for setting up external connectivity. Therefore we can examine network handling by servers and other communicating devices ranging over a wide variety of source ports to mitigate any port conflicts.

All other parameters in IP header and TCP header field were maintained same as in last test case, testing over three destination ports 80, 443 and 50000.

IP Header Fields	Value
version	4
ihl	5
tos	0
id	54321
frag	0
ttl	64

Table 3.5: IP Header Fields

TCP Header Fields	Value
sport	Randomly generated (0 to 65535)
dport	80,443,50000
flags	S (SYN)
seq	1000
ack	0
dataofs	5
reserved	0
window	8192
urgptr	0

Table 3.6: TCP Header Fields

4. **UDP broken port address experiment**

IP Header Fields	Value
version	4
tos	0
id	54321
flags	0
frag	0
ttl	64

Table 3.7: IP Header Fields

UDP Header Fields	Value
sport	Random(1024 to 65535)
dport	80,443,50000
len	None(Default)

Table 3.8: UDP Header Fields

For UDP broken source port address test case, in the IP header, the IP version is initialized as 4, IHL and total length are calculated dynamically, hence are set to default value. TOS, flags and fragment offset field are set to 0, showcasing no specific service is demanded, as well as the packets are not splitting. In the UDP header, source port is simulated to have randomness by specifying a range between 1024 to 65535 port numbers, thus ensuring novel behaviour.

Header length for UDP is set to default. Destination ports are varied across the values 80,443 and 50000. This initial configuration assigned to UDP packet helps understand how packets having variable source ports are handled. The source IP address and destination IP addresses are also command line arguments, finally with the protocol type being specified as proto = 'udp'. Checksum values are also set to default None.

5. **UDP broken length experiment**

IP Header Fields	Value
version	4
tos	0
id	54321
flags	0
frag	0
ttl	64

Table 3.9: IP Header Fields

UDP Header Fields	Value
sport	Command line variable (src_port)
dport	80,443,50000
len	Random(8 to 65535)

Table 3.10: UDP Header Fields

Broken length was initialized with a random header length field, ranging over the values 8 to 65534. All the default configuration for IP header are same as in the earlier UDP broken source port address test. Only variation, being randomized length, and source port by default expecting a command line input from the user. UDP checksum is computed dynamically by scapy. Thus network behaviour of different packet sizes can be explored, particularly bringing into limelight their effect on packet reception and channeling.

6. **QUIC random payload experiment**

IP header fields, for QUIC test, were given the same default values as in the UDP and TCP tests. Protocol number was specified to 17, which indicates UDP protocol. Source IP and destination IP address are input as command line arguments. Checksum was set to default value for dynamic calculation.

Within the UDP header, source port would be randomly initialized with any port number between 49152 and 65535, corresponding to ephemeral ports, which are used for setting up volatile communication links. Destination ports are again varied over 80, which is HTTP port, 443 which is HTTPS port and 49312, it being ephemeral port.

QUIC header, has header form as 1, showcasing a large header size. Fixed bit allocated value 1, as well as packet type being set to 0 value, indicating earliest packet. Version is 1, which highlights the edition of QUIC protocol. Packet sequence is maintained with packet number length initialised with 1 byte value, in addition to it destination connection id length and source connection id length assigned values as 8 bytes and 5 bytes respectively.

IP Header Field	**Default Value**
version	4
ihl	dynamic
tos	0
len	dynamic
id	54321
flags	0
frag	0
ttl	64
proto	UDP (17)
src	Auto-allocation
dst	Command line input

Table 3.11: Default Values for IP Header Fields

UDP Header Field	**Default Value**
sport	(49152, 65535)
dport	Command-line argument
len	dynamic
chksum	None

Table 3.12: Default Values for UDP Header Fields

QUIC Header Field	**Default Value**
header_form	1
fixed_bit	1
packet_type	0
reserved_bits	0
version	1
dst_conn_id_length	8 bytes
dst_conn_id	Random generation 8 bytes
src_conn_id_length	5 bytes
src_conn_id	Random generation 5 bytes
packet_number_length	1 byte
packet_number	Random generation 1 byte

Table 3.13: Default Values for QUIC Header Fields

3.5.3 Header modification evaluation

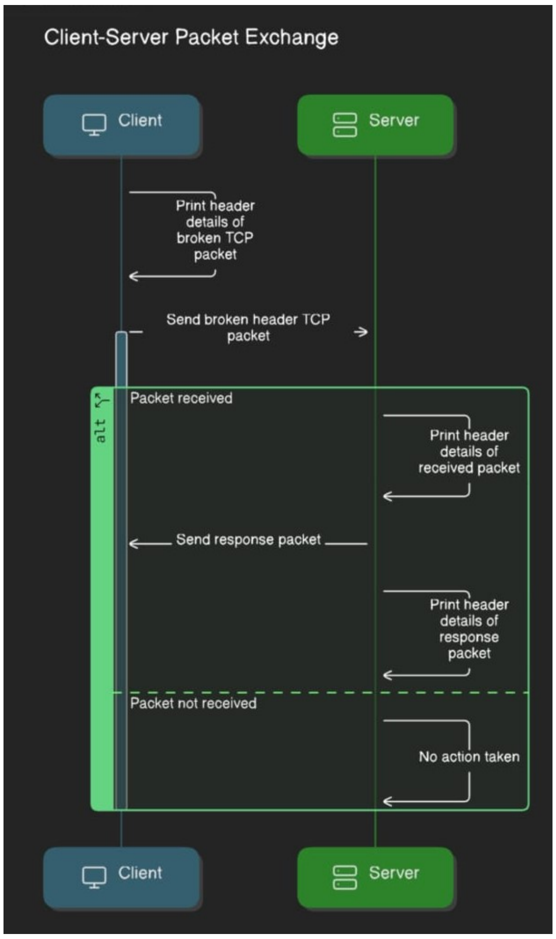

Client-Server Packet Exchange
Client
Server
Print header details of broken TCP packet
Send broken header TCP packet
alt
Packet received
Print header details of received packet
Send response packet
Print header details of response packet
Packet not received
No action taken
Client
Server

Figure 3.5: Broken TCP Client-server Setup

The Figure 3.5 is a depiction of packet exchange amongst a client-server machine pair which follow TCP protocol, with non-traditional header structure. The process commences with outgoing packets, which have been altered and sent from the client to server. Sequence numbers, data offset and source port are varied in three different versions of tests. Once the packets are sent, its header fields are printed at the client terminal, for logging of the outgoing packet header details.

Upon receipt of the packets by the server, there are two outcomes that can persist. If packets are received at the server end even after broken header structure significantly deviating from its traditional TCP transport framework , it sends back a response packet to the client.

After response packets are crafted, their header is again printed by the server to log communication. Before response packets are sent, header of the packets received at the server end is also displayed. This helped in pointing out any modifications of the deviated fields, thus resulting in better analysis of middlebox-protocol interaction.

If the packet reception fails, no response packets are sent back to the client, as all packets are blocked within the network before reaching receiver end. With this scenario, the lack of validity of standard TCP rules can be noted, thus highlighting that the packet header is not readable.

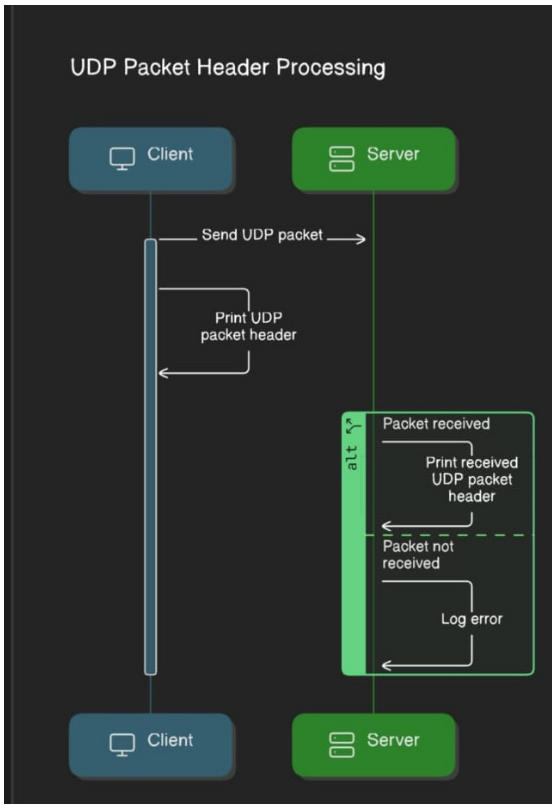


Figure 3.6: Broken UDP Client-server Setup

Figure 3.6 is the depiction of how UDP packet with altered header structure is processed amongst a client-server pair. UDP is a protocol which doesn't require a connection [14], hence the UDP packet is sent to the server without the need to establish any prior connection.

Client sends altered UDP packet , either with a random source port or broken length header field, following which the client prints header details. If the packets are received on the server side, it logs their header so that any modifications or packet drop can be analysed.In contrast to TCP, UDP doesn't have any re-transmission mechanism, leading to packet processing even if any issues are present. If packets are not received by the server, which maybe due to header being corrupted beyond a threshold, no further action is performed by the server. Server has not been programmed to send back any response to client side.

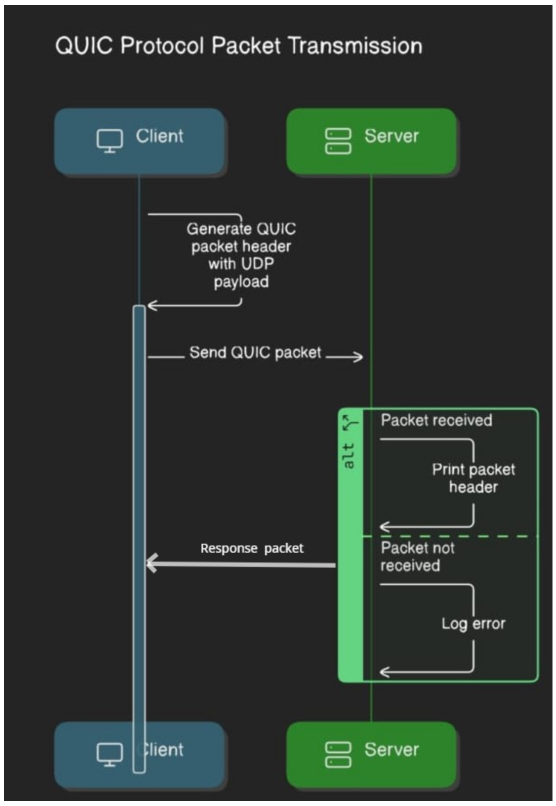


Figure 3.7: Broken QUIC Client-server Setup

Primarily, client generates a QUIC packet which is built over the UDP socket. Header specifies information about routing, and connection identification along with metadata essential for correct packet processing. QUIC data payload is encapsulated in UDP, therefore promoting data transmission with reduced latency. QUIC packet is then sent to the server , where it is checked for any modifications in its header or packet being dropped. On the successful reception of packet, the header details of the received data packets are printed, and a response packet was sent back to the client, where the header fields of response packet were judged for any variations they faced while commuting through the network.

Header log can be used to monitor the packet contents and its parameters.If no packets are received no further action is taken and an error is logged. With the tcpdump or Wireshark issues in the network traversal of the packets can be identified and better error analysis efforts can be employed.

Chapter 4

Results and Interpretation

4.1 TCP broken sequence number test

4.1.1 Observation Table

Outcome	Port 80	Port 443	Port 50000
No Change	0	0	0
Outgoing Blocked	17	17	17
Incoming Blocked	0	0	0
Partial Outgoing Block	0	0	1
Partial Incoming Block	0	0	1
Outgoing Altered	4	3	3
Incoming Altered	4	3	3

Table 4.1: Experimental Results across different ports

4.1.2 Inference:

For the experiments conducted, outcomes have been categorised in seven result types, no change indicating packets sent by the client were received with no change in header fields by the server and the server sent response packets with no change in header structure. Outgoing and incoming packets are referring to packets sent by the client to server and response packets sent by the server back to the client respectively. Packets having broken sequence number configuration for their TCP header section, were tested across 20 networks, including university wifis, cellular hotspot and public coffee shop networks. In majority of cases due to unordered packets, reassembling of packets became difficult, which could have caused the intermediate devices , i.e the network middleboxes to interpret them as malicious. This resulted in packets being blocked to meet the network security standards. There were a few exceptions, in Test 1, Test 2, Test 6 and Test 15, as documented.

In Test 1, the packets sent with destination port as 80, were completely blocked but for ports 443 and 50000 packets were received at the server end and response packets were crafted. For port 50000, partially packets crafted from the client end were blocked and were sent. At the server end, packets were received, but more than the number of packets crafted initially. Test 6, was another outlier in which packets were completely blocked. Moreover, for test 15, port 443 faced complete blockage of packets crafted by the client, with another interesting outcome observed for port 50000, where all the packets were received correctly from the client source ip, but only one response packet being sent back to the client which is not expected as number of packets crafted by client was more than one.

Amongst the packets that were successfully received and for which response packets were sent back, multiple variations in their header structure were noted. Source port addresses were changed, as compared to user specified values, besides Time to Live IP header field being decremented, signifying high possibility of routing carried out by network middleboxes. Originally even though no options were included within the TCP header packets, at the receiver end novel options like Maximum Segment Size and Timestamp were added. Source and destination port addresses were swapped, and sequence number was altered as compared to originally received packets at the server. Sometimes, sequence number for response packet were replaced by exponentially high values. This can be interpreted as a counter measure against attacks aimed at predicting the sequence numbers. Sequence numbers assigned among packets were discrete and had no continuity, indicating randomness and vague packet behaviour.

Additionally, there were also instances of data offset being increased at the receiver end in contrast to its original value. TCP header flags,were altered to "SEC" as compared to their original allocation in one of the cases. There was also a case of TCP header window size increment potentially as a measure to optimize packet flow within the network. Collectively, the results observed showcase that these modifications, were applied by network intermediaries to ensure network integrity, regulate traffic and mitigate security risks. Public and university wifis display stricter security policies into place, when compared with mobile hotspots having lower restrictions. Middleboxes thus interfered with packet traversal process, leading to packets being dropped or their header structure being changed to meet network standards, as also claimed by [13].

4.2 TCP broken Data offset test

4.2.1 Observation Table

Outcome	Port 80	Port 443	Port 50000
No Change	1	1	1
Outgoing Blocked	19	19	19
Incoming Blocked	0	0	0
Partial Outgoing Block	0	0	0
Partial Incoming Block	0	0	0
Outgoing Altered	0	0	0
Incoming Altered	1	1	1

Table 4.2: Experimental Results across different ports

4.2.2 Inference

TCP broken data offset test was conducted over 20 networks, with data offset default set to 0. In 19 of the networks tested, packets having broken data offset in their TCP header crafted by the client were blocked when sent to the server. Based on the observed block rate of 95 percent, a strong presence of security policies and mechanisms dropping abnormal packets. There was only one instance of packets crafted by the client side being received successfully at their intended destination. TCP header fields of the original packet and that being received was observed to have no alterations. The sole instance of decrement in the value of TTL field of the IP header, caused its initial value of 64 to fall down at 43, indicating traversal through network devices, resulting in this deviation of TTL field's default value.

Response packet sent back by the server to the client was accompanied with swapped source and destination port addresses in TCP header, having TTL similar to the original packets crafted by the client machine. Data offset was changed to 5, in synchronization to the standard header size equivalent to 20 bytes. However, data offset for packets received at the server remained 0, as originally assigned by the sender. Furthermore acknowledgement and sequence number fields were also changed in the response, which is analogous to sequencing and acknowledgement functionalities of TCP protocol. These outcomes were evident over all the tested ports, whereby a homogeneity can be noticed over the network's response to broken data offset field in TCP protocol header. Most of the networks tested can be concluded to possess robust procedures, which don't allow packets with non-standard configurations to pass.

Potentially the network firewalls filter out these novel header packets across all the networks evaluated, only with an exception where these packets were not interfered with and passed freely. The unchanged TCP header parameters, highlight packet resilience only with an adjusted TTL field in IP header due to middlebox effect. Server's perceived adherence to standard TCP frameworks can be seen through the data offset's correction to 20 bytes when the response packets were generated. To sum up, the consistent behaviour of network intermediary devices over diverse destination ports under analysis represent high reliability of defence mechanisms deployed over the communication channel to regulate traffic even when subjected to non-traditional transport behaviour.

4.3 TCP broken source port test

4.3.1 Observation Table

Outcome	Port 80	Port 443	Port 50000
No Change	1	1	1
Outgoing Blocked	17	17	18
Incoming Blocked	0	0	0
Partial Outgoing Block	0	0	0
Partial Incoming Block	0	0	0
Outgoing Altered	2	2	1
Incoming Altered	3	2	2

Table 4.3: Experimental Results across different ports

4.3.2 Inference

Broken source port test results revealed several different ways in which packets having non standard source ports are handled by network. In 16 out of the 20 test conducted, the packets with manipulated source ports were totally blocked and couldn't reach the receiver. This observation implies deployment of network firewalls and packet filters within the communication link, that cause packets with invalid source port addresses to get dropped. Total blockade of these packets in over 80 percent of total experiments outcomes points to necessity of preserving source port's integrity and any violation of these norms can lead to packet traffic getting rejected.

Some exceptions could be noticed where packets with changed source port field configuration were able to reach the server, accompanied by generation of corresponding response packets. For test 1, packets initially crafted were blocked when destination ports were set to 80 and 50000,although for port 443 they were received successfully. As compared to initial header field values, the received packet fields exhibited several modifications. Original source port address which was set to 12364 was changed in the received packets, along with data offset increasing from 5 to 6.

Moreover TCP options like Maximum Segment Size were introduced and TCP header length was increased. Furthermore TTL in IP header was decreased besides addition of padding in the TCP header. In Test 2, there were no alterations in the header structure of packets received at the server. Even though packets were blocked for port 50000 in test 6, their header structure was changed over port 80 and port 443.

Similarly, as far as test 15 was concerned packets on port 443 ceased to reach the destination, but packets with destination ports as 80 and 50000 showed modifications like change of original source port number, changes in sequence number and addition of TCP options, coupled with window scaling. Response packets consistently varied when viewed against received packets, having swapped destination and source port addresses, modified sequence number and acknowledgement parameters.

Modification of TCP header fields signifies that the network perform a normalization and sanitizaton of packets to make them comply with standard network regulations. Reconfiguration of header fields suggests an attempted correction by the network intermediary devices to handle non-traditional traffic, minimize risks and avoid imposing an outright packet blockade, by employing a more sophisticated approach allowing better management of non-conforming packets.

4.4 UDP Broken source port test

4.4.1 Observation Table

Outcome	Port 80	Port 443	Port 50000
No Change	2	1	2
Packets Blocked	17	18	17
Partially Blocked	0	0	0
Packets Altered	1	1	1

Table 4.4: Experimental Results across different ports

4.4.2 Inference

Based on the UDP broken source port test results, majority of packets were blocked before being received. Block percentage across port 80, 443 and 50000 was 85, 90 and 85 percent, demonstrating robust firewall or packet filter mechanisms. The criticality of HTTPS traffic on port 443 makes it more alert against anomalous packet behaviour, although in tests 1, 2 and 15 the packets were not blocked indicating flexibility in restrictions or lack of pack anomaly detection capabilities. In some cases, due to the change in source port in UDP header and TTL field in IP header, packet structure became more cautious, highlighting intermediate network filtering before their receipt at the server.

The source port modification in some cases can be attributed to potential usage of port randomization techniques or changing header as an implementation of security measures. Subsequently, the idea that middleboxes can allow some packets with novel header to pass, but with modified fields can be inferred from the outcomes analysed.

4.5 UDP Broken Length test

4.5.1 Observation Table

Outcome	Port 80	Port 443	Port 50000
No Change	1	1	1
Packets Blocked	19	19	19
Partially Blocked	0	0	0
Packets Altered	0	0	0

Table 4.5: Experimental Results across different ports

4.5.2 Inference

The test results for broken UDP header length field experiment revealed that 19 out of 20 networks discarded the novel header packets across all the destination ports. With over 95 percent block rate for the UDP packets crafted at the sender, the importance of header length's consistency can be concluded. Header field length is a decisive factor in influencing proper packet integrity and reassembly, with direct impact on packet processing. There was an instance of UDP packet passing through the network and reaching the server pointing out possible vulnerability in the network infrastructure. Majority of tests involved packet blocking, thus supporting the fact that networks are well equipped for handling packet anomalies.

4.6 QUIC Random payload test

4.6.1 Observation Table

Outcome	Port 80	Port 443	Port 49312
Passed	15	15	15
Outgoing Blocked	4	4	4
Incoming Blocked	7	7	8
Packet Altered	1	1	1

Table 4.6: Experimental Results across different ports

4.6.2 Inference

QUIC's test performance indicate high adaptability and robustness even in cases where UDP and TCP novel header protocols can underperform or collapse totally. Over the course of 22 tests carried out, the QUIC packets having randomized UDP payload fields were able to pass through the network both as initial packets crafted by sender to the server and from response packets sent by the server to the client machine. QUIC's higher efficiency in packet transmission, establishing strong connectivity and improved capability in manipulating data flow even in adverse network systems can be easily deduced.

QUIC faced some instances of packets meant to be received at the server side getting blocked and 7 cases of response packets being dropped while transiting through the communication link. These numbers are lower in contrast broken TCP and UDP transport. Despite the randomness of the UDP payload in QUIC packet architecture, there was only a single instance of header field modification, conveying QUIC's higher resistance against congestion and interference that maybe introduced in the network ecosystem.

These conclusions are inline with the study by[17], where QUIC's high potential to replace traditional TCP and UDP has been brought into limelight. The novel configurations made in QUIC's traditional header, didn't much affect its efficiency, and thus even after the randomization of UDP payload fields, the novel QUIC behaviour can be utilised as a potential alternative over a limited scale. Therefore we can draw the final outcome that QUIC significantly outmatches TCP and UDP in their performance and is a promising transport option to be deployed in dynamic and complicated network environments. The degree of impact middleboxes have on QUIC transport structure is far lesser than other transport protocols which can be accredited to its low latency connection, encryption of its header fields and high reliability.

Chapter 5

Summary and Future Scope

5.1 Summary

Current research is focused on the constraints and change in the behaviour of transport protocols specifically TCP and UDP. Moreover, QUIC is a novel protocol that was tested with random variations in its UDP payload to simulate a possible alternative for it,which would be analysed for its interaction with middlebox elements across different networks and check its capabilities if used in place of QUIC. Even though middleboxes are responsible for controlling the network traffic and provide security, they often produce unnecessary changes in packet header structures, which cause either the packets to be dropped completely or their orientation is totally modified. Usually, traditional transport protocols like TCP and UDP are handled well by the network intermediaries.

Our study had major emphasis on exploring how, changes in TCP header structure such as fragmenting the sequence of packets, randomizing source port allocations and completely breaking the data offset would effect their transmission ability. Conclusion drawn from these test was that relative sequence numbers, full length data offset and proper source port allocation is necessary for avoiding the packets being dropped or modified. Middleboxes introduced ossification of the transport layer even while UDP was tested with broken length and random source port addressing.

Enhanced transport APIs, and usage of middlebox-proof measures in transport layer architectures are some of the proposed mitigation strategies to counter adverse effects by network intermediate elements such as NATS , firewalls etc. QUIC has been a prominent choice to replace traditional transport protocols due to its highly efficient encryption and multiplexing characteristics. However, based on the research's test findings, there are several inconsistencies in QUIC's execution. Scapy, Wireshark and tcpdump were used to craft packets, understand the changes in header structures and finally highlight how flexible was the transport layer when novel protocols were used for data communication.

5.2 Future Scope

According to the key insights obtained from this project, multiple possibilities for future research have emerged. One of the central aspects is designing new protocols that are impervious to interference caused by middleboxes. In addition to this, there is also a significant need to equip protocols with dynamic detection capabilities, against the involvement of any middlebox within the communication channel. Profiling of middleboxes is very essential, especially including various types of networks and geographical areas. Consistency in performance of novel protocols like QUIC is crucial, as it will address problems, such as packet modification and blocking. Novel protocols can be further improved by addition of security enhancements to be less vulnerable to ossification.

Close collaboration with middlebox vendors, is very critical for the advancement of this research and draw out solutions for better communication. Overtime, novel protocols can be adopted to mitigate ossification effects. Evaluation of transport protocols, with major emphasis on user experience, several features like throughput, latency and robustness in distinct network intermediary arrangements, can display important trends.[5] has laid foundation for implementing rigorous specifications and test validation, to develop transport layer protocols. By incorporating insights from this study, protocols being tested with broken header configurations can be improved to have enhanced adaptability over diverse range of network ecosystems. Subsequently, symbolic evaluation methods cab be navigated through by the transport protocols, and understand the deviations in outcomes from their expected values. Integrating these methodologies into future work can help identify protocol vulnerabilities, when prone to altered input parameters.

Lastly, testing infrastructure can be significantly improved to enable packet transmission and reception along with a support for wide range of header configurations, helping in optimization of network, transport as well as application layers.This will result in better data transfer and flexibility even if middleboxes exist within the route being used by packets for traversal.

Bibliography

[1] Mark Allman. On the performance of middleboxes. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 307–312, 2003.

[2] Sahel Alouneh et al. Quic transmission protocol: Test-bed design, implementation and experimental evaluation. *Journal of Electrical Engineering*, 72(1):20–28, 2021.

[3] Zack West alpharithms. Transmission-control-protocol. https://www.alpharithms.com/transmission-control-protocol-tcp-252209/.

[4] Internet Assigned Numbers Authority. Protocol numbers. https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtmlInternet$_Assigned_Numbers_Authority,$2024.

[5] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Rigorous specification and conformance testing techniques for network protocols, as applied to tcp, udp, and sockets. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 265–276, 2005.

[6] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Http over udp: an experimental investigation of quic. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 609–614, 2015.

[7] Scapy Community. What is scapy? https://scapy.net/, 2024.

[8] Korian Edeline, Mirja Kühlewind, Brian Trammell, Emile Aben, and Benoit Donnet. Using udp for internet transport evolution. *arXiv preprint arXiv:1612.07816*, 2016.

[9] Rodrigo Fonseca, George Porter, Randy Katz, Scott Shenker, and Ion Stoica. Ip options are not an option. Technical report, Technical report, EECS Department, University of California, Berkeley, 2005.

[10] geeksforgeeks. User datagram protocol. https://www.geeksforgeeks.org/user-datagram-protocol-udp/, 2024.

[11] Piyush Goyal and Anurag Goyal. Comparative study of two most popular packet sniffing tools-tcpdump and wireshark. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 77–81. IEEE, 2017.

[12] Karl-Johan Grinnemo, Tom Jones, Gorry Fairhurst, David Ros, Anna Brunstrom, and Per Hurtig. Towards a flexible internet transport layer architecture. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–7. IEEE, 2016.

[13] 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.

[14] Joseph Iyanda. Connection-oriented tcp vs connectionless udp: A comparative analysis of network protocols.

[15] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols. In *proceedings of the 2017 internet measurement conference*, pages 290–303, 2017.

[16] 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.

[17] 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.

[18] Robin Marx. Http/3 from a to z: Core concepts. https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/, 2021.

[19] 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.

[20] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the evolution of transport protocols in the internet. *ACM SIGCOMM Computer Communication Review*, 35(2):37–52, 2005.

[21] Saad Mneimneh. Computer networks udp and tcp. *Hunter College of CUNY. New York*, 2008.

[22] 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. IEEE, 2018.

[23] Giorgos Papastergiou, Gorry Fairhurst, David Ros, Anna Brunstrom, Karl-Johan Grinnemo, Per Hurtig, Naeem Khademi, Michael Tüxen, Michael Welzl, Dragana Damjanovic, et al. De-ossifying the internet transport layer: A survey and future perspectives. *IEEE Communications Surveys & Tutorials*, 19(1):619–639, 2016.

[24] R Rohith, Minal Moharir, G Shobha, et al. Scapy-a powerful interactive packet manipulation program. In *2018 international conference on networking, embedded and wireless systems (ICNEWS)*, pages 1–5. IEEE, 2018.

[25] Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.