

On the Predictability of the Deployment of Network Mechanism

Naima Noor Shorna



Thesis submitted for the degree of
Master in Informatics: Programming and System
Architecture
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Autumn 2022

On the Predictability of the Deployment of Network Mechanism

Naima Noor Shorna

© 2022 Naima Noor Shorna

On the Predictability of the Deployment of Network Mechanism

Abstract

This thesis investigates the use of packet header information in trace files derived from Internet measurements at MAWI from the years 2020, 2021, and 2022 and scrutinizes these information to report the key changes in the Internet header fields over time. That is, by observing how certain header fields are used, we can determine whether or not the Internet is using newly developed network mechanisms. To do this, we parse the trace files and gather all the unusual data that was discovered in the packets to search for atypical data in the packets. Our research shows that there are no recognizable patterns in the data, confirming no significant changes in the header fields throughout the years. However, we get some exciting findings which leave room for future deep analysis, and the tool we implemented to carry out the analysis have a good chance for further development.

Keywords: traffic captures, packet header, header fields, Internet protocols, MAWI.

Acknowledgements

First, my most profound appreciation goes to my supervisors, Michael Welzl and Safiqul Islam, for providing me with this excellent opportunity to pursue this thesis under them. Their guidance, ideas, encouragement, and insightful and valuable feedback amidst their busy schedule have been invaluable during the thesis.

I want to thank my parents for encouraging and supporting me in this endeavor. I would also like to thank everyone who has supported me emotionally and intellectually throughout my master's program.

Finally, I dedicate this thesis to my beloved Matias for putting up with me. I couldn't have done this without your unwavering support!

Contents

Abstract	i
Acknowledgements	ii
1 Introduction	6
1.1 Motivation	6
1.2 Problem Definition	7
1.3 Contributions	7
1.4 Research Questions	8
1.5 Outline	8
2 Background	10
2.1 Packet Switching	10
2.2 Performance Mechanisms	11
2.3 The OSI Network Model	12
2.4 The TCP/IP Network Model	13
2.5 Internet Protocol and Network Protocols	15
2.6 Network Packets' Structure	15
2.7 IP	17
2.7.1 IPv4	17
2.7.2 IPv6	21
2.8 TCP	23
2.9 TCP/IP Suite	26
2.10 UDP	28
2.11 ICMP	29
2.11.1 ICMPv4	30
2.11.2 ICMPv6	31
2.12 Packet Header and Packet Trace Analysis	32
2.13 Packet Analyzers	33
2.14 Packet Loss and Delay	34
2.15 Anomalies in Packet	35
2.16 The Analysis of Network Traffic	35
3 Solution Approach	38
3.1 Related Work	38
3.2 Research Gap	39
3.3 Proposed Solution	41
3.3.1 Python	41

3.3.2	Libtrace	42
3.3.3	plt-libtrace Module	43
3.4	Dataset	43
3.4.1	The WIDE Project	44
3.4.2	MAWI Working Group	45
3.5	Implementation	45
3.5.1	Json Configuration	45
3.5.2	Implementation Details	48
3.5.3	Result Code	52
4	Result	56
4.1	Interesting Header Fields	56
4.2	Result Analysis	58
5	Discussion and Conclusion	65
5.1	Result Findings	65
5.2	Limitations	66
5.3	Future work	66
A	Source Code	69
	Bibliography	73

List of Figures

2.1	Open Systems Interconnection (OSI) Model	12
2.2	TCP/IP Model	14
2.3	IPv4 Header	17
2.4	IPv6 fixed header	21
2.5	IPv6 Extension Header	22
2.6	TCP Header	24
2.7	TCP/IP connection establishment	27
2.8	TCP/IP sending data packets	27
2.9	TCP/IP connection termination	27
2.10	UDP Header	29
2.11	ICMP used in IPv4 and IPv6 Packets	30
2.12	ICMP Header	30
3.1	Flow Chart of the Tool	48
4.1	IPv4 ECN values	58
4.2	IPv4 Protocol	59
4.3	IPv6 flow label	60
4.4	TCP ECE & CWR Field	61
4.5	TCP Reserved Field	62

List of Tables

2.1	IP header options	20
2.2	IPv6 Priority Assignment	21
2.3	Sequence Order of IPv6	23
2.4	ICMP Control Message	31
4.1	IPv6 next header field	61
4.2	ICMP Type	63

Chapter 1

Introduction

1.1 Motivation

The paradigm shift in the 21st century has revolutionized the Tech-industry but increased its vulnerability simultaneously. In the third generation of computers, network support capabilities were introduced, as were communication components integrated into operating systems [1]. Computerized data transmission is significantly more complex than typical human communication because of the dynamic nature and communication route. An administrator may analyze intercepted messages to determine network architecture by evaluating network performance concerns such as improperly configured devices, system faults, and network activity. The passive analysis can be of different types depending on the protocol and other specifications. It can be as simple as counting packets and examining the TCP/IP headers or real-time analysis.

Data transmission via a network or communication channel is managed by using packet headers in networking. In order to maintain a steady and rapid data flow, the control data is bundled up into the packet's header. This data typically includes the packet's source IP address, destination IP address, routing information, protocol version, etcetera, enabling the packet to reach its intended destination. Data plane forwarding devices feature fine-grained packet forwarding and verification capabilities, allowing for exact control of the forwarding behavior of network packets by verifying the source authenticity and integrity of the forwarded packet [2]. Depending on the protocol, like TCP, UDP, ICMP, etcetera, headers have different formats. These formats may not be unique. Usually, one or more fields are added or are absent. Thus, indicating that these header fields have undergone many changes over time. Analyzing trace files can provide insight into the changes header fields have undergone, and a pattern can be determined from the data.

Even though the Internet was first created as an experimental packet-switched network, many of its features are now considered "set in stone" [3]. In addition, Internet ossification makes it almost impossible to deploy

new protocols and extensions on the Internet, making transport layer evolution harder. For this purpose, the header fields are to be evaluated to see if any rarely used protocol techniques have changed or if any new mechanism has evolved. Furthermore, the observed anomalies can be converted into a graphical or visual representation to develop trends for better understanding. Finally, these findings can be utilized to build sniffing tools and software in the future.

1.2 Problem Definition

While working on a system and encountering a network issue, one must maintain a high-level network and automate data and traffic monitoring. The analytic tool enables one to obtain consistent traffic statistics across several OSI layers without slowing down the network. With the aid of data packets, it is possible to extract vital information regarding health and performance that may be used to troubleshoot performance and track down anomalous data packets. The header fields depending on the protocol, like IPv4, IPv6, TCP, UDP, and ICMP, are analyzed for various purposes. Unfortunately, the headers do not maintain a consistent nature all the time.

To understand a bit more, think of a case where we want to inspect a TCP header field. First, we need to find the TCP header skipping any preceding headers, decode the header and then read the header field we are interested in. Finally, continue with the next packet. However, the case becomes complicated when it is a non-TCP packet or truncated before the TCP header. These problems can alter the complete analysis if not taken care of properly. Before handling the unfamiliar values in the header field, actions might be required to keep things in line. For example, some packet header fields require bit-shifting to achieve the correct value. So, the program may require the functionality to do right or left bit shift operations before bitwise operations, depending on the circumstances. All these steps are necessary for the smooth transmission and receiving of the data.

1.3 Contributions

In this research, innovative and efficient methods are utilized in order to retrieve the necessary information from the header. During another research work, we developed a tool utilizing the python-libtrace (plt) module to trace TCP connections. During this thesis, we developed a tool modifying the last tool and added JSON configuration to make the tool more dynamic in nature. The tool can utilize both python-libtrace (plt) module and the JSON script to gain direct access to the protocol's header fields within the packet, thus giving an upper hand over other available traffic analyzing tools. Furthermore, by utilizing python-libtrace, which provides a class inheritance hierarchy, we have a substantial advantage over the programs that handle encapsulated packets because those programs have the potential to benefit from the hierarchy. It is also

possible to generate Traces and Packets with the help of the libtrace API. Moreover, we define a list of interesting header fields in various protocols, which are analyzed to articulate a trend based on their behavior and to propose future expectations from it.

1.4 Research Questions

The research questions for the study "On the Predictability of Network Mechanism Deployment" are listed below:

How much have packet header fields changed over time?

1.5 Outline

The following is the structure of the rest of this thesis:

Chapter 2 - Background: This chapter describes the background information on the most relevant concepts behind the methods implemented in this thesis.

Chapter 3 - Solution Approach: This chapter discusses related works and gives an overall idea of what is implemented in this thesis.

Chapter 4 - Implementation: This chapter shows what has been done in this thesis. We conclude with a description of the implementation.

Chapter 5 - Results: The test results with analysis after the implementation in Chapter 4 are shown in this chapter.

Chapter 6 - Discussion and Conclusion: Finally, in this chapter, we begin by summarizing how we responded to the study questions, discuss the overall result and then conclude by suggesting possible future research prospects.

Chapter 2

Background

This chapter provides a concrete background for the development of the work. It presents a summary of the technical and theoretical frameworks related to this thesis. Besides this, the abbreviations used are also elaborated.

2.1 Packet Switching

Packet switching refers to a networking device's capacity to process packets independently of one another. It also implies that packets might follow multiple network paths to the same destination as long as they all arrive there. At the links' inputs, the vast majority of packet switches employ store-and-forward transmission. As each hop forwards a packet, it first stores it, then sends it on its way. Packets can be dropped at any hop for a variety of reasons, making this strategy extremely useful. There are multiple ways to get from one place to another. Source and destination addresses are included in each packet and are used to guide it through the network on its own. Packets from the exact file can follow a different path. If a path is congested, packets can choose any other route possible over the present network if they so choose. The transmission rate of a link is measured in bits per second, and different links can send data at varying rates. As the result of packet-switched networking, no physical wires are to be established on a network's infrastructure. Nonlinear, non-time-slotted, and irregular networks, like Ethernet, can use multiple packet connections that end at a switch. Alternatively, sequential links could use frames to begin and finish packet transfers at predetermined times. Studying packet switch architectures is also convenient with time-slotted links [4]. Nodes would switch packets based on their destination address instead of delivering each message as a single piece of data. To make a smart routing choice, a node must know the entire network and its immediate surroundings. Each node has a certain amount of memory to store this data. As a result, the routing protocol must carefully select which information about the network and neighbors to keep and which information to discard. The Internet was developed with time. It is the worldwide system of computer networks that links billions of machines together. On the Internet, these

gadgets are called hosts. The endpoint nodes are linked together by a communication infrastructure or any form of physical media.

2.2 Performance Mechanisms

Network performance depends on queuing, traffic management, prioritization, and scheduling. Meanwhile, some performance techniques include control over resources and quality of service (QoS). The regulation of resources, the establishment of SLAs, and the guarantee of service quality are all performance metrics. The individual, group, or network-wide performance can be improved by identifying traffic flow categories and measuring their temporal features with these procedures. Besides this, Capacity, latency, and RMA are all examples of performance parameters that may be included in a service. Each characteristic is essentially a name for a specific class of qualities. These qualities include bandwidth, throughput, goodput, etcetera. Delay includes variable, end-to-end, and round-trip delays. Reliability, maintainability, and availability are often referred to as RMA qualities.

Differentiated services and integrated services are two common types of QoS for IP-based traffic, which are meant to provide two different viewpoints of network services. DiffServ prioritizes QoS by supporting aggregated traffic flows per hop based on traffic behavior, while IntServ prioritizes QoS by supporting individual traffic flows throughout the network. Different from individual traffic streams, aggregated traffic flows are the target of DiffServ because network architecture and design evaluate how much memory network nodes need to store and keep state information for each unique flow. Resource consumption is unscalable due to linear network growth. Identifying and categorizing communications facilitates storing and keeping critical state data. State information includes network flows and connections' configuration, and maintenance [5]. When it comes to resource distribution, IntServ describes the values and methods used along the entire flow's path. Therefore, support for a flow at every network node along a flow's entire course is critical for IntServ's operation.

Service level agreements, on the other hand, are legally binding contracts between a service provider and an end-user that spell out the terms under which the provider will be held liable for failing to meet its obligations. While traditionally, SLAs have been deals between firms and clients, the concept can be transferred to the business world. The client should outline their expectations for each service individually. Depending on the provider, this may differ. Often, customers want the highest possible levels of performance. In theory, this is understandable, but it may be costly in practice. The service provider may argue that service levels should be purposely low to ensure that the service can be supplied at a competitive price. Alternatively, it is all a matter of judgment, and the customer will have to

analyze each service level carefully based on its commercial importance. For example, an online service’s capacity to be available at all times is a top priority for most companies because the customers depend on it. Other services, which may not be as critical, can have lesser service levels established for them.

2.3 The OSI Network Model

Part of what is recognized as network architecture is the process of developing a comprehensive, top-down framework for the network. This encompasses the interactions that exist both inside the main architectural components of the network and between those components. Some examples of these relationships include addressing and routing, network administration, performance, and security. The next step in building our network is network design, which is essential for integrating needs and traffic flows.

The OSI model has two layers: the upper and lower ones. When it comes to applications, the upper layer of the OSI model is all about software implementation. In terms of end-user interaction, the application layer is closest to the user [6]. End-users and application layers are both involved in software development life cycles. This is the layer that is directly above the one that came before it.

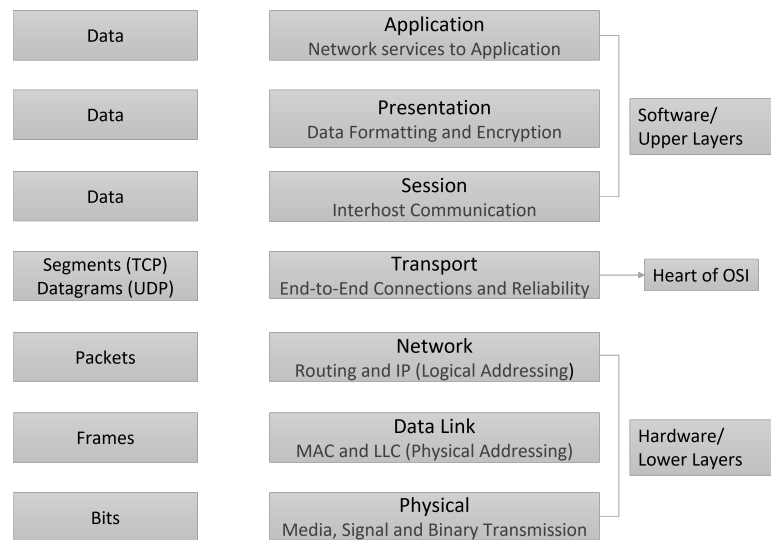


Figure 2.1: Open Systems Interconnection (OSI) Model

The OSI model’s lowest layer focuses on data transfer. The data link layer, the physical layer, and the application layer are all implemented using a combination of hardware and software. At each successive layer of the communication exchange, one or more network protocols are put into operation. Lower-layer protocols are included in the Internet protocol

suite, as are typical applications like e-mail, terminal emulation, and file transfer. The physical layer is the lowest level in the OSI model and has the closest relationship to the underlying media. The information is primarily placed on the physical media by the physical layer.

A layered architecture gives us the ability to talk about a clearly delineated section of an extensive and complicated system. This simplification in and of itself is of significant benefit since it provides modularity. Adjusting the layer's service performance is more effortless. Even if a layer's implementation updates, the rest of the system can stay working as long as it uses the services provided by the layer beneath it and provides the identical service to the layer preceding it. Network designers use layers to arrange protocols and the hardware and software that implements them in order to give structure to the design of network protocols. According to the networking paradigm, there are several layers of network infrastructure where every layer depends on the layers underneath it and supports the layers above it [7]. Similarly, protocol suites offer comparable features. The application is taken as a starting point and starts from the bottom up. Each layer, like each protocol, has a defined job. One way to characterize a layer is by the functions it provides to the layer over it, known as the layer's service model. It utilizes the services provided by the layer underneath and carries out various tasks inside that layer. A protocol is an agreed-upon collection of rules and practices for transmitting and receiving information via a network. Depending on whether transport protocol is used in conjunction with IP, packets are handled differently after they arrive at their destination. At several layers of the OSI architecture, protocols attach packet headers. The most important benefit of the OSI model is that it helps organize one's ideas regarding networks and provides beginners, journeymen, and experts with a shared vocabulary for discussing computer networking.

2.4 The TCP/IP Network Model

The TCP/IP model is one approach to simplifying the OSI architecture. The OSI model has seven layers; however, this one only has four. There are various tiers, the most basic TCP/IP Model shown in figure 2.2 is the Application Layer, followed by the Transport Layer, the Internet Layer, and finally, the Network Access Layer.

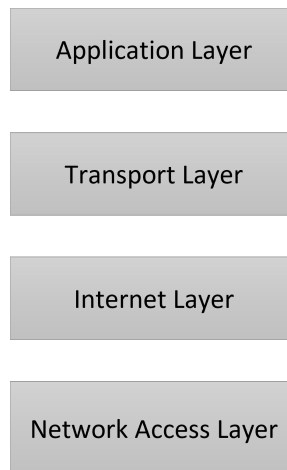


Figure 2.2: TCP/IP Model

From an OSI viewpoint, the network access layer is functionally equivalent to the combination of the Data Link and Physical layers. It ensures that hardware addressing is taken care of, and the protocols that are included in this layer make it possible for data to be physically transmitted. OSI's network layer has several similarities to the Internet layer. Protocols like IP, ARP, and ICMP govern the logical transport of data across the entire network. This model has various protocols, and the protocol, Address Resolution Protocol (ARP), determines a host's hardware address. There are several forms of the Address Resolution Protocol signal, including reverse, proxy, gratuitous, etcetera.

The transport layer of the TCP/IP model is equivalent to the OSI model's transport layer. End-to-end communication and error-free data transfer are tasked to it. It acts as a barrier between the data and the higher-level applications. It can sequence and break data effectively. It has a method for flow control and the acknowledging functionality to regulate data transmission. It is a robust protocol, but the latency it takes is high. Costs rise as a direct result of increased inefficiency. The two main protocols residing at this layer are TCP and UDP. Within the TCP/IP model, the application layer is the topmost layer. The Application Layer combines functions from the top three layers of OSI: Presentation, Application, and Session [8]. In addition to controlling the user-interface specifications, it manages communication between nodes. Some standard protocols used in the application layer are HTTP, FTP, Telnet, SSH, SMTP, NTP, and DNS.

This model has many advantages and can be used for various purposes. It is an industry-standard model that can be used to solve practical networking issues as it is interoperable, allowing cross-platform communication between heterogeneous networks. Any individual or organization can utilize it because it is a set of open protocols which any specific institute does not hold. It can be said that it is a client-server design that is scalable. As a result of this, it is possible to install new networks without disrupting

current services. Each machine on the network is given an IP address, allowing it to be identified by the network. It also assigns a domain name to each site.

2.5 Internet Protocol and Network Protocols

Protocols govern Internet interactions between autonomous computers. In a protocol, two or more communicating entities establish the format and order of messages exchanged, and the actions are taken when a message or other event is transmitted or received. Endpoint congestion control techniques manage data packet transmission speeds, while router routing procedures determine a packet's path from origin to destination [9]. The Internet Protocol Suite is the most popular protocol of OSI because it can be used to interact across various networks, from a single computer to a vast area network. The Internet Protocol can also be used to route data packets between networks as the internet packets carry data. Routers can send packets to the correct destination because of the IP information included in each one. IP addresses are assigned to every device or domain connected to the Internet, and packets are routed to those addresses. The communication process is broken down into separate tasks at each layer of the OSI model by the protocols used by the networks. Besides this, routing devices have typically been termed gateways in the Internet world. The Internet's routers are arranged in a hierarchical structure. Using routers, data can be routed over a series of networks controlled by the same administrative organization. A variety of protocols known as "internal gateway protocols" are utilized by routers for information sharing within autonomous systems. The external gateway protocol is used by routers that transport data between autonomous systems. These routers are known as exterior routers. Routing protocols in IP are dynamic. When a route is calculated dynamically, the software on the routing devices must do it frequently. Instead of dynamic routing, static routing relies on the network administrator to set up the routes, which remain in effect until the administrator changes them. Destination addresses and next hops are the building blocks of an IP routing table.

2.6 Network Packets' Structure

The transport-layer segment is made up of the application-layer message as well as the information of the header of the transport layer. As a result, the message of the application layer is incorporated into the transport layer section. Some examples of this supplemental data are detection of deviation bits that let the receiver to determine if bits in the information have been modified during transport and information that allows the transport layer of the recipient to transmit the message to the application. This information may also include error-detection bits. After the segment is delivered from the transport layer, the network layer generates a data

packet on the network layer by adding details to the header of network layer, such as the addresses of the end system devices i.e., source and destination addresses. After that, the datagram is delivered to the link layer, which will generate a link-layer frame by including its very own link-layer header information before passing it on. This allows us to see that a packet has two types of data, a field for header and a field for payload, at every layer it traverses. A packet from a higher layer is often what makes up the payload. The header, the data, and the trailer are the three components of a network packet. The packet's header specifies how to handle the information contained within it. The following are examples of possible instructions: [10]

- The internet protocol address;
- Header;
- Size of the payload;
- Packet length;
- Imprecision diagnosis by checksum;
- Offset-based fragmentation for restoration of the address;
- A packet's utmost intermediary nodes;
- Source;
- Sequence number;
- Different flags to tell a router to break a packet;
- Protocol being used;
- Data packet details and mechanisms to keep networks in sync.

A router determines in which route to send a packet with the help of an IP address. IP packets are processed independently, meaning packets from the same application or link may be routed through different channels [11]. The term "payload" is commonly used to describe the information contained within a packet. In a nutshell, this is the information that the packet must convey in order to reach its intended recipient. Blank data is frequently used to pad the payload in order to fit it into a fixed-length packet. The header size has nearly doubled in Internet Protocol version 6, which will have a significant impact on bandwidth. Payload Header Suppression is a bandwidth saving technology that removes the redundant packet overhead.

From section 2.7 to section 2.11, all the protocols we are interested for this thesis work are described briefly.

2.7 IP

The Internet Protocol (IP) is the core protocol used by the Transmission Control Protocol or Internet Protocol suite. Internet Protocol (IP) is in charge of exchanging information and messages between computers and other devices on the same or different networks linked to the web. For example, the data is transmitted from a sending party to a receiving party server by checking the packet headers for the IP addresses of the respective domains. Internet Protocol (IP) addresses are unique numerical identifiers used by devices to transfer data across a network or connect to another device over the Internet. Besides this, every IP address comprises two parts: the network information and the host information.

IP defines the packet structure because network protocols divide each message into numerous packets. A header and data are included in each IP packet. The header contains all routing information, such as source and destination addresses, that aids packet routing. In contrast, the data/payload is either the actual data content or a portion of a higher-level transport protocol. As a result, IP is also known as a routable protocol.

Two types of IP versions exist: IPv4 and IPv6.

2.7.1 IPv4

In developing the Internet Protocol (IP), IPv4 is the fourth version. It is one of the foundational protocols of outcome-based networking systems used on the Internet and other packet-switched networks. The Internet Protocol version 4 (IPv4) is the primary protocol to direct Internet traffic. Only 13 of IPv4's 14 fields are essential; the others are either discretionary or seldom-used alternatives. For example, depending on the options field size as shown in 2.3, the IPv4 header might be 20–60 bytes. IPv4 is being used by the vast majority of websites.

Version	IHL	Service Type	Total length	
Identification			Flags	Fragmentation Offset
Time to Live	Protocol		Header Checksum	
Source IP Address				
Destination IP Address				
Option				
Data				

Figure 2.3: IPv4 Header

The following are various fields of IPv4 Packet header:

- **Version:** All internet protocol packet headers start with a 4-bit version number. For IPv4 the field is always set to 0100 which indicates 4 in binary.

- **IHL:** IHL or Internet Header Length is the length of the entire IP header, which is 4 bytes in size. This field displays 32-bit header words. For instance, a proper header must be of the value that is 32-bit multiplied by the realistic and possible minimum or maximum value of a header.
- **DSCP:** The requested service type is coded in a 6-bit parameter called the Differentiated Services Code Point, previously known as "Type of service." For example, it provides real-time data streaming or VoIP (Voice over IP). Networks can also use this field to define how to handle a datagram during transport. Priority bits represent the first three bits of a datagram.

0	1	2	3	4	5
Precedence			D	T	R

Precedence:

Value	Description
0	Routine
1	Priority
2	Immediate
3	Flash
4	Flash Override
5	CRITIC/ECP
6	Internetwork Control
7	Network Control

D – 1 bit: 0 = Normal Delay; 1 = Low Delay

T – 1 bit: 0 = Normal Throughput; 1 = High Throughput

R – 1 bit: 0 = Normal Reliability; 1 = High Reliability

- **ECN:** The 2-bit Explicit Congestion Notification field indicates data congestion in a path without deleting the data. If the network supports it and both destinations activate it, RFC 3168 allows this added capability [12].
- **Total Length:** This 16-bit value specifies the total length of the IP packet, along with the IP header and IP data. Bytes are used to measure the length. The header can be as small as 20 bytes (without data) and as large as 65535 bytes. Hosts require datagrams of 576 bytes in length and should be able to process it. However, newer hosts can process packets of greater size.
- **Identification:** The 16-bit Identification field aids in the identification of IP datagram fragments. This field's value is exclusive to the source-destination pair and protocol when the datagram is in transit across the Internet. It has been used to add information to packet tracing in

an experimental manner. RFC 6864, on the other hand, prohibits such use [13].

- **Flags:** It is a 3-bit field that helps to identify and control fragments. For instance, if an IP packet is too large to process, these flags indicate whether it can be fragmented.

Bit 0	Reserved (R), must be set to zero
Bit 1	Don't Fragments (DF)
Bit 2	More Fragments (MF)

DF - Control the fragmentation of the datagram.

Value	Description
0	Fragment if necessary
1	Don't fragment

MF – Indicates if the datagram contains additional fragments

Value	Description
0	This is the last fragment
1	More fragments follow this fragment

The last fragment differs from unfragmented packets by having a non-zero fragment offset field.

- **Fragment Offset:** This field indicates the number of bytes in the datagram prior to the specified section, expressed as the number of 8 bytes of frames. The initial offset is a zero offset. A maximum offset of 65528 bytes is possible with the 13-bit field.
- **TTL:** The IP packet's lifespan is controlled by this 8-bit field. Every packet is sent with some TTL values to avoid looping in the network. TTL values are decremented by one each time a packet traverses an intermediate hop, and once they reach 0, the packet is considered obsolete and is deleted. The value of TTL can be 0 to 255.
- **Protocols:** The 8-bit Protocol field denotes which protocol has been used in the data portion. The protocol numbers 1, 6, and 17 respectively represent ICMP, Transmission Control Protocol (TCP), and User Datagram Protocol (UDP).
- **Header Checksum:** To ensure the header is error-free, this 16-bit field is used. A router verifies the integrity of a packet by checking the checksum field in the IP header. The packet is discarded if somehow the respective values are different.

- **Source Address:** In IPv4, this is the 32-bit address of the source of the packet.
- **Destination Address:** This is a 32-bit IPv4 address of the destination of the packet.
- **IP options:** This field is used if the internet header length is greater than 5. Nonetheless, this field is frequently overlooked. The size of this field varies based on whether there are zero or many selections. The options field format can take one of two forms.
Case 1: An option-type octet.
Case 2: There are interestingly three different fields for the option-type. Each byte contains information about the option's type, length, and data.

0	1	2	3	4	5	6	7
C	Class		Option number				

C - On disintegration, the Copied flag specifies whether or not this option should be replicated in all child pieces.

Value	Description
0	Don't Copy
1	Copy

Option classes are:

Value	Description
0	Control
1	Reserved for future use
2	Debugging and measurement
3	Reserved for future use

Table 2.1 shows a few examples of options.

Number	Class	Length	Description
0	0	1	End of Option list
1	0	1	No Operation
2	0	11	Security
3	0	var	Loose Source Routing
4	2	var	Timestamp
8	0	4	Stream ID
9	0	var	Strict Source Routing

Table 2.1: IP header options.

- **Data:** This field stores data from protocol layer.

2.7.2 IPv6

Each IPv6 packet comprises addressing control information as well as user data in the payload. Besides this, IPv6 is on the rise since the quantity of IPv4 addresses are far less than the number of users. A fixed header and optional extension headers make up the control information in IPv6 packets. The length of the permanent header is 40 bytes long. The IPv6 header as described in RFC 8200 [14] are shown in Figure 2.4.

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
Source IP Address			
Destination IP Address			

Figure 2.4: IPv6 fixed header

- **Version:** This field identifies the version of the Internet protocol. It is a 4-bit field that carries the value six or binary value 0110.
- **Traffic class:** The 8 bits traffic class field indicating priority of IPv6 packet can be broken up into two distinct sections. In order to classify packets, a Differentiated Service (DS) field is utilised, and it makes use of the 6 bits that are most relevant defined in RFC 2474 [15]. The least significant two bits are used to store the Explicit Congestion Notification (ECN) information as defined in RFC 3168 [12]. There are two different kinds of traffic, namely congestion controlled traffic, and uncontrolled traffic. At present, traffic class only uses four bits, with 0 - 7 corresponding to congestion controlled traffic and 8 - 15 corresponding to uncontrolled traffic [16].

Priority Value	Description
0	No Specific traffic
1	Background data
2	Unattended data traffic
3	Reserved
4	Attended bulk data traffic
5	Reserved
6	Interactive traffic
7	Control traffic

Table 2.2: IPv6 Priority Assignment

- **Flow label:** The creation of this massive 20-bit field is for the purpose of identifying packet flow in a sequential order. To determine whether or not a given packet is part of a particular information flow, a unique flow label must be applied to the packet in question. If a packet does not belong to any flow in the network, it will have the

unique flow label i.e., label 0. Flow Labels are described in RFC 3697, which provides details on their use [17].

- **Payload length:** This 16-bit unsigned numeric field contains the packet's octet payload length. Any extension headers are included in the Payload length but not the main IPv6 header. Besides this, using the Jumbo payload option in the Hop-by-Hop Extension header allow payloads above sixty-five thousand bytes. The large packets are referred to as jumbograms in RFC 2675 [18]. They are commonly found in supercomputers, and data centers run at very high speeds.
- **Next Header:** The 8-bit "next header" field can be used in two ways. Without an extension header, it functions similarly to the one present in IPv4. If not, it will tell you what kind of header comes after the IPv6 one.
- **Hop limit:** This field is composed of 8 bits and is equivalent to IPv4's TTL. Each packet sender node decreases values by one. The packet is dumped when the counter reaches zero.
- **Source Address:** This source address field is 128 bits and stores the unicast address belonging to IPv6 protocol of the source of the packet.
- **Destination Address:** This destination address field is 128 bits and holds the unicast or multicast IPv6-protocol address of the packet's intended recipient.
- **Extension Header:** The shortcomings of the IPv4 Option Field prompted the development of Extension Headers, which were subsequently incorporated into IPv6. The extension header comes between the fixed header and the protocol header of the higher layer. Where the upper layer protocol header is in charge of relaying any extraneous data about the network layer to the user [19]. Each Extension Header has a value that is completely unique to it.

In the case where extension headers are being used, the next header value of an IPv6 protocol's fixed header will refer to the first extension header being used. The next header field of the first extension header will lead to the second extension header, and vice versa if additional extension headers are present. Besides this, the header of upper layer is indicated in the final extension header's next header value. As a consequence of this, in the shape of a linked list, each of the header points to the following one. Figure 2.5 illustrates how the sequence of IPv6 extension header looks like.

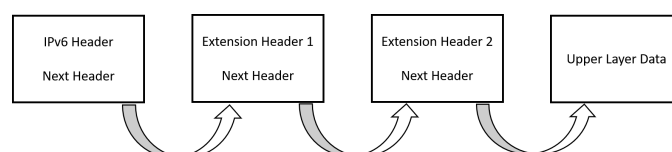


Figure 2.5: IPv6 Extension Header

Table 2.3 shows the sequence of Extension Headers.

Order	Header Type	Next Header Code	Description
2	Hop-by-Hop Options	0	Examined by all devices on the path
3	Destination Options (with Routing Options)	60	Examined by destination of the packet
4	Routing Header	43	Methods to take routing decision
5	Fragment Header	44	Contains parameters of fragmented datagram done source
6	Authentication Header	51	Verify authenticity
7	Encapsulating Security Payload	50	Carries encrypted data
8	Destination Options	60	Options that need to be examined only by the destination of the packet
9	Mobility Header	135	Parameters used with Mobile IPv6
-	No next header	59	-
Upper Layer	TCP	6	-
Upper Layer	UDP	17	-
Upper Layer	ICMP	58	-

Table 2.3: Sequence Order of IPv6

2.8 TCP

TCP or transmission control protocol is a connection-oriented protocol of the transport layer that allows a reliable exchange of messages between application programs and computing devices over a network. Due to this factor, a connection must be established between the sender and receiver before any information can be transmitted. Besides this, the reliable connection is to be maintained until the data has been exchanged. It uses a three-way handshake technique to develop a valid connection between the sender and the receiver (active open). During the data transmission pro-

cedure, TCP divides large data into numerous small packets to ensure data integrity.

TCP enables one-to-one connectivity at the Transport layer, which means we can't exchange data with more than one device simultaneously. In short, TCP can't send messages to numerous network recipients. In this case, UDP (User Datagram Protocol) might be utilized instead. TCP increases latency while ensuring data reliability through handshakes, retransmission of missing data, and waiting for out-of-data. As a result, TCP is inconvenient for real-time applications; instead, UDP is preferable, while it is less dependable.

TCP Header: TCP segments are formatted as data in IP packets. A TCP segment contains a Header and data section. There are ten mandatory fields in the header, as well as an optional extension field. The header can vary in size on the basis of the options field's size and range from twenty and sixty bytes.

Source Port		Destination Port	
Sequence Number			
Acknowledgement number			
Data Offset	Reserved	Flags	Window
Checksum		Urgent Pointer	
Options			

Figure 2.6: TCP Header

- **Source port:** This field specifies the sender's port number in 16 bits.
- **Destination port:** This is a 16-bit field that specifies the receiver's port number.
- **Sequence number:** The sequence number field contains 32 bits that represent the amount of data transferred during a TCP connection. For a new TCP connection (three-way handshake), a random 32-bit value is used as the initial sequence number. This sequence number is used by the receiver, who responds with an acknowledgment. If the segments are not received in the correct order, this is used to reassemble the message at the receiving end.
- **Acknowledgment number:** The receiver uses this 32-bit field to request the next TCP segment. This value will be the sequence number incremented by one. It's a confirmation that the previous bytes were received correctly.
- **DO:** Data offset (DO) field or header length has 4 bits. It signifies the length of a TCP header using 32 bits words, allowing us to determine when the actual payload begins.

- **RSV:** This 3-bit reserved field is always set to 0 because it is unused.
- **Flags:** This 9-bit flags field consists of six original 1-bit flags known as control bits and three additional flags, including one experimental flag defined in RFC 3560 [20] and two congestion control flags defined in RFC 3168 [12]. These flags are used to initiate connections, send data, and terminate connections.
 1. NS (1-bit): ECN-nonce - concealment protection; An optional field to prevent the TCP sender from accidentally or maliciously concealing marked packets [21].
 2. CWR (1-bit): Congestion Window Reduced - Whenever a TCP segment forwarded by the sending node includes the ECE bit, the CWR flag is set.
 3. ECE (1-bit): Explicit Congestion Notification (ECN)- Echo. TCP connections only use this flag to indicate ECN capability from the sending node.
 4. URG (1-bit): When this bit is set for a packet, that means the packet should process with any latency over all other packets.
 5. ACK (1-bit): This bit is used for acknowledgment upon receiving an SYN packet.
 6. PSH (1-bit): The PSH flag is used to facilitates immediate data transmission without any buffering.
 7. RST (1-bit): Setting the reset flag results in the abrupt termination of an open connection.
 8. SYN (1-bit): This 1 bit is used to set the initial sequence number (ISN) and for initiating a TCP connection.
 9. FIN (1-bit): A TCP connection is terminated using this finish bit. Because TCP is a full-duplex protocol, both sides must utilize the FIN bit to terminate the connection. This is the standard way of terminating a connection.
- **Window size:** The maximum number of bytes that the receiver will accept is determined by the 16-bit window field. It is used to inform the sender that the receiver requires more data than is currently available. It accomplishes this by specifying the number of bytes in the acknowledgment field that are not included in the sequence number.
- **Checksum:** 16-bit checksum is used to determine whether the TCP header is valid.
- **Urgent pointer:** This 16-bit urgent pointer field is used to identify where the urgent data terminates when the URG bit is set.
- **Options:** This field is optional and can range from 0 and 320 bits. The data offset field determines the length of this field. There are three fields available for each option: 1 byte of 'option kind', 1 byte of 'option length', and variable 'option data' field.

2.9 TCP/IP Suite

TCP and IP are two different protocols, but TCP is mostly used when coupled with IP to guarantee accurate and correct relay of data to its intended destination within a network. The IP part is responsible for moving of data packets between nodes, on the other hand TCP is responsible for verifying the delivery of packets. The initial objectives of TCP and IP are: [22]

- If a network fails or in case of a node failure, connectivity must still be possible.
- There has to be a wide range of communication options available over the Internet.
- The underlying design of the Internet must be able to: Support multiple network topologies.
- Allow for decentralized control of its resources.
- Use resources efficiently.
- Allow easy host connectivity.
- Internet infrastructure resources must be traceable.

TCP/IP has become the fundament of the Internet. So today, the Internet Protocol Suite is also referred to as TCP/IP. However, there are other related protocols as well in the suite. In networking, TCP/IP Suite use a four layered framework that is analogous to the OSI model's 7 layers [23].

Below is a brief discussion of how a packet is transmitted over TCP/IP [24].

Step-1: Connection Establishment

Using three-way handshake, a client established a connection with the server.

1. **SYN:** The client initiates communication by sending a packet to the server with the SYN flag set to 1. The client initiates the series by setting sequence number to a chosen arbitrary number.
2. **SYN-ACK:** The server responds with a 'synchronized acknowledgement', both flags set to 1. Then the server sets the acknowledge number to 'random number+1' and the sequence number to another random number.
3. **ACK:** The client replies with an ACK or an acknowledgement. The acknowledgement number is set to the received sequence value incremented by 1. The three-way handshake only uses control packets, which generally include no payload. Once the connection is established, the actual data transfer is started.

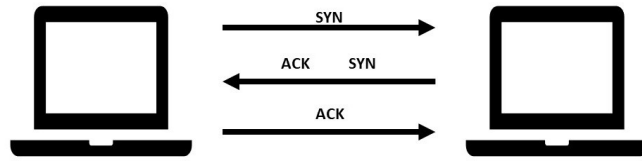


Figure 2.7: TCP/IP connection establishment

Step -2: Send data packets

It is crucial that the recipient always acknowledges what they have received. So, when a packet is sent with data and sequence number, the recipient sends back the ACK to the sender and increases the acknowledge number by the length of the received data.

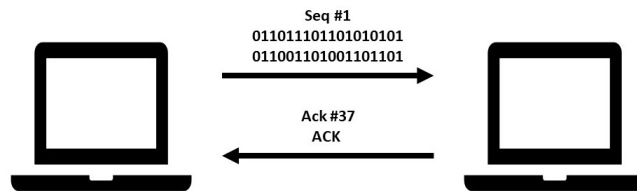


Figure 2.8: TCP/IP sending data packets

The acknowledge number is the next byte of the sequence number the receiver expects to receive. The sequence number in a packet is always valid, but the acknowledge number is only valid when the ACK flag is set to one.

Step -3: Closing connection

Connections in TCP are full-duplex. Each side of the connection can independently terminate the connection. Thus, if one end of the connection wishes to discontinue the connection, it will send the other end a FIN flag. Usually, TCP uses a four-way handshake to close the connection from both endpoints. There can be two scenarios here:

1. **Full close:** The client (or Server) sends the FIN packet. The Server acknowledges it with an ACK, and then Server also sends a FIN packet. Then the client ACK's the Server's FIN. Consequently, in order to terminate a TCP connection, both ends must exchange FIN and ACK combined.

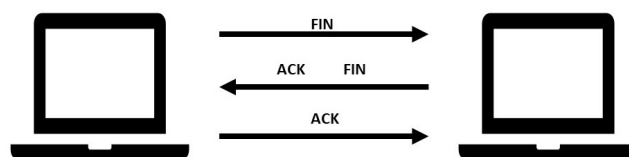


Figure 2.9: TCP/IP connection termination

2. **Half close:** This happens when one endpoint closes the connection but the other does not. The Client (or Server) sends a FIN, and the Server ACK's the FIN.

Problems with packets exchange: Several packet-based messaging issues can be resolved through TCP, including lost packets, out-of-order packets, packet duplication, and corrupted packets.

- **Detecting lost packet:** The sender sets a timeout and places the packet in a resend queue after sending. If the sender does not receive an Acknowledgment from the receiver before the timeout period ends, the packet is resent with a new timeout threshold set to double the original value. But if the packet was actually not lost but just taking a slower route to arrive or be acknowledged, then re-transmission can lead to data duplication. Then the recipient can simply use the sequence number to discard the duplicate packets.
- **Out of order packets:** Using the sequence number and acknowledge number, TCP can detect out-of-order packets. If the receiver gets a sequence number that is greater than the sequence number it has acknowledged, it will respond with an acknowledgment with the projected sequence number. It's possible that the lost packet was simply delayed in transit or that it was never sent at all. In either scenario, though, the receiver can use the sequence number to put the data back together in the proper order.

2.10 UDP

The UDP or User Datagram Protocol is a simplified protocol for the Internet's transport layer that is used to transfer data packets. The transmission of data with UDP relies on a straightforward connectionless approach that employs a minimal protocol mechanism. Transactions are the focus of the protocol, and there is no assurance that messages will be delivered in the correct order or that duplicates will be avoided [25]. If the application doesn't need reliable transport, UDP is the more resourceful protocol to utilize. Furthermore, it is beneficial in situations when data only flows in one direction and recognition of the data is not important. UDP is extremely important to the operation of streaming applications like VoIP and multimedia streaming.

UDP Header: When compared to TCP, which can have a header anywhere from twenty bytes to sixty bytes in length, UDP's is fixed at eight bytes and extremely straightforward. UDP datagram headers consist of 4 fields of 2 bytes each. Together, these 8 bytes encompass all necessary header information [26]. After the header is the data section, which contains the payload data for the application.

Source IP Address	Destination IP Address
UDP Length	UDP Checksum
Data	

Figure 2.10: UDP Header

- **Source Port:** The Source Port field is 16 bits in length and is used to identify the port number of sources. Since its use is entirely discretionary, if this field is left blank, the default value will be 0 [25].
- **Destination Port:** This field is 16 bits in length and is used to identify the port of the destined packet.
- **Length:** This field is 16 bits in length and determines the length of UDP, including the header and the data.
- **Checksum:** Checksum field is 16 bits in length and is the sum of the pseudo-header of IPv4, the UDP header, and the data.

2.11 ICMP

Every IP module implements the Internet Control Message Protocol (ICMP) to diagnose network communication issues at the network layer. ICMP as of IP datagrams, transmits host-specific information regarding network difficulties. The most common usage of ICMP is to relay error messages, but it can also provide operational information on the IP protocol layer's configuration and IP packet disposition.

ICMP is different from other transport layer protocols like TCP or UDP, as this protocol usually does not exchange any data between network devices. Also, ICMP does not provide IP with any level of reliability [27]. The Internet Control Message Protocol (ICMP) can be exploited by malicious actors in the network to perform a variety of attacks, including flood attacks, pings of death attack, and other methods that can compromise essential system operations and expose their network configuration.

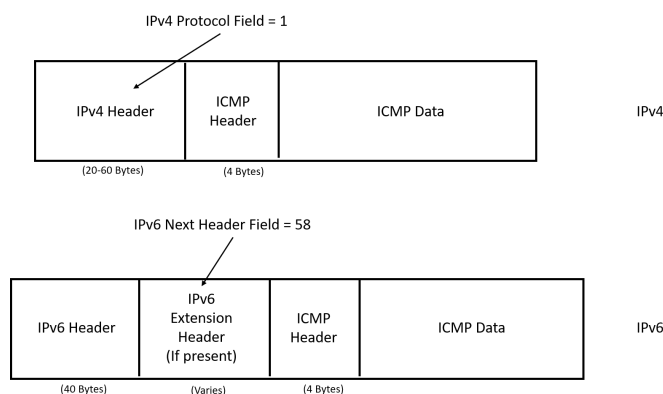


Figure 2.11: ICMP used in IPv4 and IPv6 Packets

2.11.1 ICMPv4

In IPv4, if the value of the protocol field is 1, then it indicates that the datagram carries ICMPv4. There is an IPv4 header followed by an ICMPv4 header. An ICMP packet begins with a header of 8 bytes and then carries information of varying sizes. There is a fixed format for the first 4 bytes of the header, but the last 4 bytes are determined by the ICMP packet type/code [28].



Figure 2.12: ICMP Header

ICMPv4 Header:

- **Type:** The type of message is represented by this 8-bit field. The remaining data's format can be determined based on the value. The Type field in ICMPv4 is designated for 42 distinct values that identify the specific message. Only around 8 of them are used on a daily basis.
- **Code:** The Code field is used in a variety of ICMP messages. This field is used to provide additional details about the message's meaning. It is a 8-bit field.
- **Checksum:** The 16-bit Checksum field covers the entire ICMPv4 message.

Data-32bit: ICMP error messages contains a copy of the IPv4 header and the first 8 bytes of the packet that generated the error. ICMP error messages can be up to 576 bytes long. The host uses this data to match the message to the relevant process.

Control Messages: The value in the Type field is used to identify control messages. The message's code field provides further context information. Since the protocol's introduction, some control messages have been deprecated. Some example of common control messages are given in Table 2.4.

Type	Code	Status	Description
3 - Destination Unreachable	0		Network unreachable
	1		Host unreachable
	2		Protocol unreachable
	3		Port unreachable
	4		Fragmentation required and DF flag set
	5		Source route failed
5 - Redirect Message	0		Redirect Datagram for the Network
	1		Redirect Datagram for the Host
	2		Redirect Datagram for the ToS and network
	3		Redirect Datagram for the ToS and host
11 - Time Exceeded	0		TTL expired in transit
	1		Fragment reassembly time exceeded
12 - Parameter Problem: Bad IP header	0		Pointer indicates the error
	1		Missing a required option
	2		Bad length

Table 2.4: ICMP Control Message

2.11.2 ICMPv6

Each ICMPv6 message begins with an extension header of IPv6 and may begin with 0, one, or more IPv6 extension headers. The ICMPv6 header can be recognised from the previous header by looking for the next header value of fifty-eight [29].

ICMPv6 headers are essentially identical to ICMPv4, except that in ICMPv6, the checksum also includes a pseudo-header derived from the IPv6 header.

Errors in packet processing are reported using ICMPv6, and other internet-layer services like diagnostics are performed using ICMPv6 (ICMPv6 "ping"). Because it is an essential feature of IPv6, every IPv6 node must implement the full ICMPv6 base protocol, including all messages and behaviours specified below. Error and information messages are the two

types of ICMPv6 messages. The entries in the message type field are distinguished by errors by setting the highest order bit to 0. The types of messages for error messages range from 0 to 127, whereas message Types for informative messages range from 128 to 255.

The destination and the source Addresses of IPV6 must be recognized by the node that transmits the message of ICMPV6 prior to the calculation of checksum. It is imperative that the node choose the source address of the message if it has more than one unicast address. Node's unicast IP address must be the source of the ICMPv6 packet. The address of the destination of the packet, the source address should be selected in the same way as the source address would be for any other packet generated by the node. There are other ways to select an IP address that is reachable from the destination of an ICMPv6 packet, but this is not mandatory. Reducing the bandwidth and transmission cost of ICMPv6 error messages produced by the node of IPv6 is a top priority. This could happen if the originator of a flood of incorrect packets ignores the ICMPv6 error messages that result from sending them. When the transport protocol fails to find a receiver for a packet, the node at the endpoint should send back a message about the out of reach with a code 4 and inform the source. Error situations may or may not be resolved in the near future, depending on the specific situation. It is therefore possible that an ICMP error message's response may be affected by a variety of factors, including the time at which the error message is received as well as previous knowledge of the network error conditions that are being reported, as well as an understanding of how the receiving host operates in its current network scenario.

2.12 Packet Header and Packet Trace Analysis

Using packet tracing, one may check that a packet followed the correct route across all layers before arriving at its endpoint. The absolute control impact can be accomplished among adjoining junctions by regulating these vital throughways. Depending on the level of traffic, we can employ a variety of control methods. Even basic scientific tasks, such as calculating the traffic observed on a particular port of TCP, can necessitate complex algorithms in order to decrypt the packet headers and derive information from the basic binary code. In packet header processing, the output link and quality of service are defined by the packet's route identification and categorization [30]. That is why most packet trace analysis is done with the aid of a software library that can decode and handle collected packets in an abstract form. Most commonly, libtrace, a library that can read packet traces in the PCAP format, is used. Libtrace overcomes the shortcomings of libpcap and other trace-processing libraries by reading, processing, and publishing packet traces [31]. A more brief discussion about Libtrace is given in section 3.3.2.

2.13 Packet Analyzers

As a critical error detection approach in information security, packet analysis can recreate even the entirety of traffic flow at a given moment, assuming the collected packet information is adequate. Concerns like which protocols can be understood, which is the ideal platform to use given the level of experience, and which packet sniffer would function best for the network environment must be addressed before a packet analysis can begin. Packet analysis involves many internal processes and exchanges at various levels. When a packet goes through a node, all network data is received by the node's 'network interface card,' but all the hardware and addresses are ignored in permissive mode. However, traffic-filtering switches complicate network data collecting, and the kernel passes control to the OS kernel once a network interface card copies a packet into driver memory [32]. Each packet that passes over a network can be intercepted using a packet sniffing technique. Packet sniffing is a method by which a user examines the data of other network users. On both switched and non-switched networks, it runs perfectly. Packet sniffers are versatile tools that can be used either for administrative or malicious reasons. It is dependent on the goals of the user. An attacker may use packet sniffing's several applications, such as breach detection and system administration, to gather sensitive information such as passwords, IP addresses, and current protocols [33].

Packet capture and visualization are the primary objectives of network packet analysis. Some packet sniffers like 'WireShark' are user-friendly, and the user can go through the packet details quickly. The packet-header details window in Wireshark includes information about the packet specified from the packet list. An IP datagram and Ethernet frame are included in this packet's detailed information. The amount of Ethernet and IP-layer information presented in the packet details window can be modified by clicking on the right or down-pointing arrowheads next to each line of the Ethernet frame or IP datagram. Each Ethernet frame or IP datagram line's arrowheads on the packet details window can be used to change the amount of Ethernet and IP layer detail shown in the window. In addition, TCP or UDP information about the packet's transmission is shown if it was sent using one of those protocols. Besides this, Wireshark has a capability that allows it to distinguish between different types of communications based on their protocol. Wireshark's color codes are helpful when studying packets. Three distinct types of network traffic are represented: DNS, UDP SNMP, and HTTP. In Wireshark, one can choose from various color schemes that can be customized.

Sniffing methods can be divided into three categories. The first method of packet sniffing is IP-based. In the IP-based sniffing method, all packets matching the IP filter are sniffed when the network card is in promiscuous mode. When the IP address filter is enabled, it is set to capture all packets regardless of their source. Only networks without switches can make use of this technique. While the network card is in promiscuous mode,

the MAC-based sniffing method can sniff all packets that meet the MAC address filter. Lastly, the ARP-based sniffing method used a different approach. In this method, the network card does not need to activate in a permissive state due to the statelessness of the ARP protocol. Because of this, sniffing is still feasible on a network switch.

An administrator of an operational network can use data gathered from network monitoring to take proactive measures to keep it running well and to provide users with usage statistics. Some metrics used by network administrators include link activity, error rates, and link status. This data can be used in the long run to see and forecast growth and detect and replace a faulty component before it entirely fails. Detailed and thorough records of Internet traffic across a link are provided by packet traces, which are commonly utilized in Internet measurement data. They cannot be handled and analyzed with simple scripting or statistical languages since they are collected and kept exactly as they occurred on the network. The user can send out packets with an incorrect address. Machines running sniffers will receive the packets; therefore, we can assume they are sniffers.

2.14 Packet Loss and Delay

During any communication we do on the internet, packets carry data over networks. Packet loss occurs when these communications are dropped, or distorted [34]. In the chain of servers, packets face a variety of delays as they move from server to server. These delays can occur at any node along the path. As a result of network congestion and packet delivery delays, packet loss occurs, which prevents the video decoder from properly decoding the video stream and directly impact the video quality [35].

The total nodal delay consists of processing, queueing, transmission, and propagation delay [36]. It is necessary to analyze the packet's header in order to determine where it should be sent, which accounts for a part of the delay in processing. Other factors can affect the processing time, such as the time it takes to check for bit-level errors made during the transmission of the packet from the upstream node to the router. The amount of time required to perform this check is typically on the order of microseconds or less in high-speed routers.

As the packet waits to be transmitted onto the link, it is subjected to a delay known as queuing delay while it is located in the queue. The time a packet spends in the queue depends on how many other packets have arrived since it entered the queue and are waiting to be sent across the link. For example, if there is no other packet sent at the same time as ours and the queue is empty, then the amount of time our packet spends in the queue will equal zero.

The transmission latency is the time it takes to send every bit of data in

a packet through the network. Transmission delays tend to fall somewhere on the order of microseconds to milliseconds. After a bit has been put into the link, it must then propagate to the router to whom it is being sent. Data travels from the link's origin to the router in the propagation delay. Bit propagation matches link speed, and then the transmission medium determines link propagation speed. Packet loss has a variety of effects on different applications. For example, a 10% packet loss might add only one second to a ten-second data download. Delays can become more significant if there is a higher packet loss or latency rate. Especially vulnerable to packet loss are real-time applications like speech and video. Even a two percent packet loss might cause the discussion to become stilted and incoherent to the average listener or spectator.

2.15 Anomalies in Packet

Any deviation from a network's usual data transmission pattern is known as an anomaly. Trojans and cyber threats are just two examples of anomalies; incorrect information packets and changes in transmission due to network issues, delivery delays, and malfunctioning hardware. Network operators need help to detect and diagnose threats due to a large amount of network flow and reduced network infrastructure efficiency caused by Anomalies. Newer anomaly identification methods use empirical network analysis to find new or unexpected anomalies [37]. When it comes to network operators and end-users, identifying abnormalities is essential. However, this is challenging to solve due to the vast amount of high-dimensional, unpredictable data.

Not all packets are helpful or safe, and network communication is not always secure. Malicious network traffic data packets are designed to compromise or overwhelming a network. This can take the form of a DDoS assault, exploitation of a vulnerability, or a variety of other cyber-attacks. Traffic anomalies must be recognized quickly and accurately to maintain network health; otherwise, these can pollute networks and hosts with false data [38]. High-speed file transfers are critical to large-scale scientific procedures. Quality factors like guaranteed bandwidth and no packet loss or data duplication are required for these transmissions. Successful file transfers require procedures such as thresholds and statistical analysis to detect aberrant patterns. Network administrators who use their knowledge and experience to make judgments on how to diagnose and remedy problems regularly monitor and analyze network data.

2.16 The Analysis of Network Traffic

Reading data packets sent over a network is known as network sniffing. Sniffing can be accomplished with the use of specialized software tools or hardware. Sniffing can be used to grab login passwords, eavesdrop on chat messages, or capture files while they are being transmitted across

the network. The machines communicate with each other by broadcasting messages over a network using IP addresses. Computer and networking administrators continuously monitor system traffic to maintain network health, safety, and growth. This analysis has become complex with open-source technologies due to the amount of data transported by today's networks. Network administrators need tools that monitor and evaluate traffic aggregations with headers and payloads [39].

The device monitoring application can be created to use its distinct network using wireless technology, which allows it to operate independently of the customer's underlying wired network. Each of the large networks that make up the Internet is in control of a block of IP addresses, and these networks collectively are known as autonomous systems. Packets are routed across ASes depending on their destination IP addresses using several routing protocols, including BGP. Routing tables on routers specify which ASes packets should travel through to reach their desired destination as rapidly as possible. Each AS takes responsibility for a different IP address, so packets travel amongst them until they reach a single AS with that address. Those packets are subsequently routed internally by the AS to their final destination. Using the packet sniffer, we can capture and analyze traffic. And then, reports are generated based on traffic analysis. Protocol-based filtering is also implemented for a variety of protocols, such as TCP, IP, and UDP. The system generates an alert when suspicious activity is detected.

Chapter 3

Solution Approach

In this chapter, we discuss related works and provide a general overview of what is done in the thesis. Last but not least, it discusses the methodology used in this thesis.

3.1 Related Work

Certain tools have been developed for the convenience of communication between two endpoints. These tools have different features and specifications depending on the services required. 'BruteShark' [40] is a cross-platform, open-source network forensics tool. It can harvest passwords, display a network map, reconstruct TCP sessions, and convert encrypted password hashes to Hashcat format for offline Brute Force attacks. Security researchers and network administrators will benefit from this project because it enables them to analyze network traffic more efficiently. At the same time, the tool helped them to look for vulnerabilities in the network that would make it possible for an attacker to gain access. 'eCAP' [41] is a software interface for outsourcing content analysis and adaptation to a loadable module to be used by network applications, such as HTTP proxies and ICAP servers. But 'eCap' is an outdated platform that does not incorporate new protocols and technologies. Thus, leaving a large room for improvement.

Data Plane Development Kit (DPDK) [42] is another collection of libraries and drivers for handling large amounts of data in a short amount of time. It was created to run on a variety of processors. It started with the support for the Intel x86 and has since added international business machine power, Advanced RISC Machine (ARM), etcetera. Linux is the predominant operating system to run it. A FreeBSD port is available for several DPDK features. Whereas, 'IPsumdump' [43] is a system for transforming TCP or IP dump files into a ASCII format. IPsumdump reads packets from network interfaces, tcpdump files, and existing ipsumdump files. It transparently decompresses tcpdump or ipsumdump files when needed. It may randomly sample traffic, filter it by masking IP addresses, and organize packets from separate sources by using timestamp. besides this, it can also

create a tcpdump file with packet data.

Furthermore, 'tcptrace' [44] is a tool developed at Ohio University by Shawn Ostermann for analyzing TCP dump files. It can read files created by tcpdump, snoop, etherpeek, HP Net Metrix, and WinDump, among other common packet-capture programs. Tcptrace can display total time, bytes, and fragments given and collect, re-transmissions, round trip regulations, window ads, outputs, and more for each identified connection. It also generates graphs for further research. Tcpflow [45], another tool, records TCP connection details and saves them in a format that makes protocol study and troubleshooting easy. A program such as 'tcpdump' [46] displays an overview of the packets observed on the wire, but it does not typically store the data in the process of being delivered. On the other hand, tcpflow will recreate the raw data streams and save each flow in its own individual file so that it may be analyzed later. However, it has the capability to optionally isolate pcap flows on a per-tcp-flow basis for granular analysis.

The 'fling' [47] tool helped others learn how verified protocols and UDP encapsulations worked across Internet channels. A fling test specification has two files: json and pcap. Sending info to fling's endpoints. Fling customers download and perform tests from fling servers using HTTPS/TCP. In case of failure, the json file defines a three-time communication sequence (using pcap packets). These iterations helped to discover 'random' disruptive behavior, like a drop caused by a short interruption and sometimes congestion. If three packets are dropped, the test fails. Adding tracebox [48] to traceroute identifies middlebox interference along nearly every route. Tracebox transfers IP packets with different TTLs and examines the ICMP responses to calculate a TCP packet's TTL. Tracebox can detect upstream middlebox tampering since current routers include the entire IP packet in the ICMP transmission. Tracebox can also detect the network hop where middlebox interference occurs. In addition to this, the end results show unusual middlebox types on an internet-connected testbed.

3.2 Research Gap

Analyzing the data as it travels over the network can be done by using packet sniffing. A sniffing tool is an excellent way to do that. It can be used for a variety of tasks, including traffic monitoring, analysis, and troubleshooting. Packet sniffers can intercept passwords, usernames, and other private data. A packet sniffer can be improved in the future by implementing new features. Thus, sniffing should be done in a way that improves network performance and security. Both the network operations and research groups benefit significantly from the usage of network packet traces. Full packet traces have been captured using various tools and software libraries, such as libpcap [46].

Libtrace [31], a new open-source software library that attempts to improve both usability and speed over libpcap, was designed in response to the many shortcomings of libpcap. It is possible to construct portable trace analysis tools using libtrace's expanded programming API without having to worry about specifics such as file compression or intermediate protocol headers. When performing I/O-bounded analytic activities, libtrace uses threaded I/O and caching techniques to enhance speed even further. Libtrace-based programs can use any available capture format. In other words, they can be run on any supported capture source without the need for any additional code. Reading from a PCAP trace file is no different from reading from a live capture interface or Endace DAG card [49]. Besides this, network packet debugging may be done quickly and easily by using the library called libpacketdump. Stdout will be flooded with the data from a libtrace_packet_t, which will be parsed for any layers it is aware of (such as Ethernet or TCP). Newer protocols can easily be supported because they were designed as modules. Moreover, TracepktDump, a libpacketdump-based alternative to tcpdump, can be used to gather packet data. All these features make the tool developed in this thesis of dynamic nature and unique as compared to the already developed sniffers.

The packet sniffers are of various kinds depending on the desired outcome of the network. The utilization of network layer protocols differs from network to network. IPV6 addresses are used to identify and track machines on the internet. For the internet to function, each device connected to it must have its unique IP address. Still, most of the networks use orthodox IPv4. Moreover, some of the packet sniffers do not read the ICMP and complicate the procedure. Nevertheless, with this tool, in addition to ICMP, our application will be capable of supporting dynamic protocol versions IPV4, IPv6, and TCP. This feature makes this a valuable tool in various fields deploying different types or models of the networks that could be OSI network model or TCP network model. Because of this, the tool becomes user-friendly. By making even a minor change to the JSON script, one will generate the result that we get to alter. Despite this, we need to make some modest adjustments to our tools in order to update the TCP option fields and the UDP header.

This research utilizes unique and efficient methods to get the desired fields of the header. Instead of using the JSON script to gain access to the TCP option number included in the packet, the python-libtrace (plt) module is directly utilized. This will allow us to determine which TCP option number is contained in the packet. As the name suggests, JSON is an abbreviation for "JavaScript Object Notation," a messaging format derived from the JavaScript programming language. In today's world, JSON messages can be transmitted in any text format and are supported by almost every computer language but this only makes it beneficial in some conditions. In some scenarios, JSON does slow processing, which could be harmful to the network analysis. Using python-libtrace gives us a clear edge as it is a set of rules and tools to define and exchange these messages rather than

just a message format. Programs that deal with encapsulated packets can benefit from the class inheritance hierarchy provided by python-libtrace. It is possible to produce Traces and Packets with libtrace's API, which is a collection of functions for reading and writing packets, as well as extracting 'decodes' for various portions of the packets, such as IP (IPv4) and TCP and UDP headers. In addition to this, a set of Python classes called python-libtrace (plt) gives Python access to the libtrace objects they need. There are methods in each class that enable access to libtrace functions and fields in libtrace decodes. Only the Trace class procedures are used to describe the trace itself; the Trace object reads or writes python-libtrace packets. One should use higher-level techniques in one's methods if one wants to work with TCP objects. That way, Python will not have to garbage-collect an IP6 object, for example, during the method's execution.

The different studies and research papers reviewed by the researcher are very inspiring and thought-provoking. The studies related to this topic were focused on mechanisms that lacked the wholesome approach toward networking and packet sniffing. Anomaly detection and other techniques lack the base for improvement and further development. With this, it could be said that large-scale flow capture is becoming increasingly realistic, opening the door to traffic analysis tools that may detect and identify a wide range of irregularities. Nevertheless, the difficulty of adequately evaluating this enormous data source in order to diagnose anomalies has yet to be met. In light of the preceding discussion, the research gaps were successfully identified. There needed to be more data available for the packet anomalies and handling various protocols simultaneously. The study aims to fill this research gap and explore the various future aspects of advancement in computer networking.

3.3 Proposed Solution

The Trace processing and analysis can be accomplished using any number of libraries and tools, but this tool is different on various levels and deploys diverse techniques. Subsequent are the unique dimensions of this tool.

3.3.1 Python

Executing a pcap programmatically can come in handy in some circumstances. Hence, in such cases, a custom program will be extremely helpful in parsing the pcaps and yielding the relevant data points. For our thesis, we are analyzing the network traces using Python.

Python is used in this tool as it interprets code line by line. Errors stop execution and are reported. It only shows one mistake if the program has numerous. This simplifies bug-hunting. Python framework that makes working with online or offline network data simple and easy to understand by

providing data structures that are quick, flexible, and expressive. It aspires to be the primary high-level building element for doing realistic network data analysis in Python in the real world. In addition to this, its overarching objective is to evolve into a standard framework for the analysis of network data used by researchers, so ensuring the consistency of data across different trials. Furthermore, Python is cross-platform. It allows developers to work on one device and implement on another without modifying code. It supports Windows, Mac, and Linux and can be distributed without an interpreter on any OS. Python can develop executable programs for popular platforms.

3.3.2 Libtrace

Our tool is based on libtrace. Libtrace is an open-source software library by the WAND network research group from The University of Waikato [50]. The C programming language was used to create Libtrace.

In contemporary practice, there is no artificial measurement of traffic is used to analyze network behavior. Two key steps can be identified here: capture, which involves pulling information from the network, and analysis, which involves using some measure to evaluate the information. Users of the libtrace API can link the protocol header for every layer on or around the transport layer without intermediary headers. Both layer-specific and protocol-specific header extraction utilities exist.

The same libtrace program can be used with all supported capture formats; there is no difference between offline and live capture formats, there are advantages for development, analysis programs can be tested offline before being put into live operation, protocol layers are directly accessible, and so on. In addition, the following trace formats are supported and can be read from or written to: legacy ATM trace file, PCAPNG trace file, PCAP interface, PCAP trace file, DAG live capture and ERF trace file. Legacy Ethernet and POS trace files, ATM cell header files, etcetera are all examples of read-only trace formats. Besides this, Libtrace is capable of reading and writing compressed trace files natively. lzo, bzip2, Gzip, etcetera are the compression formats that are supported (lzo supports is write only). Compression and decompression on a separate thread, libtrace allow file operations to be performed much faster. This is very helpful for analysis tasks where input and output performance is a bottleneck. It simplifies tractable assessment and eliminates the issue of repetitive code. Also, libtrace takes care of it properly, so we do not have to worry about managing special scenarios like Internet protocol segmentation, malformed headers, etcetera. Libtrace programs can read and write any supported trace format without user code changes or recompilation because the API is 'capture format agnostic'. The application adapts to command line inputs for trace format and placement. Libtrace can also automatically recognize various input sources capture formats, so there is rarely a need to specify the capture format [31]. Since the main program does not have to wait around for a packet to arrive,

it is helpful that the libtrace API's trace event method supports concurrent processing from initial data providers. Libtrace's API will produce a count-down if packets are not present. This is extremely useful when making a graphical user interface for a monitoring application.

The supported headers are Ethernet, 802.11, VLAN (802.1q), MPLS, PPPoE, IPv4, IPv6, ICMP, ICMP6, TCP, UDP, OSPF [31].

3.3.3 plt-libtrace Module

A Python module known as python-libtrace (plt) [51] gives us the ability to work with packet trace data by utilizing the libtrace library that is provided by WAND. It is not meant to be a straightforward translation of the libtrace calls from C into Python; instead, it is designed to give a clean, straightforward, and Python-like method of dealing with libtrace. For the purpose of decoding protocol headers, for instance, the field names from the RFCs are utilized rather than the names provided by libtrace. In addition, programs that deal with encapsulated packets can benefit from the class inheritance hierarchy provided by python-libtrace. In a python-libtrace program, **trace()** is used to create a trace object, and then we can open the trace simply by using **trace.start()** and **trace.close()** to close the trace file. Below we are listing some methods we have used in our tool from the plt-libtrace module other than the ones mentioned above:

- **start_output()** - open a trace file for writing.
- **close_output()** - closes a trace file.
- **write_packet()** - Writes the data from a Packet to an OutputTrace.
- **pkt.ip** - gets an IPv4 header from the packet.
- **ip.src_prefixip.src_prefix** - gets an IPv4 source address.
- **ip.src_port** - gets an IPv4 source port .
- **ip.pkt_len** - gets an IPv4 total packet length.
- **ip.hdr_len** - gets an IPv4 header length.

By using plt-libtrace, packet headers can be examined directly at the metadata, the link, the IP, and the transport layers.

3.4 Dataset

As part of our thesis research, we analyzed traffic statistics collected by the Measurement and Analysis of the Wide Internet (MAWI) Working Group Traffic Archive. However, in the beginning, we considered two different datasets, the CAIDA and MAWI datasets. CAIDA, widely used in many graphical representations and modeling tools, enables the smooth development of software. Moreover, while preserving the confidentiality of the consumers and institutions that give data or network access, CAIDA

collects multiple types of data in regionally and topologically dispersed sites and makes this data available to the academic community [52]. All the traffic collections are not public in CAIDA and lacks daily trace collection. On the other hand, the MAWI dataset has a vast collection of daily traces which are easily accessible, making it a perfect fit for traffic anomaly detection. So, due to the higher efficiency level coupled with the availability of a wide range of internet traffic traces, MAWI is selected for the analysis. And we used samplepoint-F, which has been running since 2006. This is because samplepoint-A was ceased in 2000, and samplepoint-B was discontinued in 2006 and substituted by samplepoint-F. The MAWI Working Group of the WIDE Project is responsible for this traffic data repository [53].

3.4.1 The WIDE Project

We are approaching an era of qualitative transformation for Internet technologies as we observe the end of an era of rapid deployment and growth and the beginning of a new phase. The WIDE Project has been around for thirty years providing significant impact in all aspects of network environments. The WIDE Project's fundamental premise is to provide a worldwide connection not just between computers and humans but by connecting everything to each other. This project intends to create a highly public information infrastructure that will perform a beneficial role from an individual and societal standpoint, as well as bring to the forefront the related difficulties and challenges that must be addressed in order to make this a reality [54]. The "WIDE Internet" also serves as a location for scientific research and testing. The findings of the research are incorporated into operational procedures to create a more effective network environment for the benefit of society and the commercial sector.

Working groups steer the research activities of the WIDE Project. Through collaboration with a range of other areas, each of the working groups fosters research activities and identifies new research subjects. A robust network is required to convey the massive quantity of data created in our daily lives to the rest of the globe. As a result, the Network Operation Infrastructure group is working on enhanced Internet operating technologies through R&D (Research and Development). MAWI is one of the research working group for the Network Operation Infrastructure which is based on Traffic Measurement and Analysis [55]. It is a shared research base that links individuals participating in separate projects together. The WIDE Internet is a network that allows users to connect with one another in order to share and discuss information on a regular basis. It is run through a process that involves repeated trial and demonstration.

3.4.2 MAWI Working Group

Since the start of the WIDE Project, the MAWI Working Group has been measuring, analyzing, evaluating, and verifying network traffic. Their aim is to bring together what they've learned from network operations and evaluate the study findings against the actual traffic. In addition, by studying the behaviors of the network, they are able to determine whether the network performs as intended or learns from unexpected outcomes. However, considering how challenging it is to perform measurements at the site of operation, the MAWI Working Group was formed with the goal of focusing on measurement and evaluation. Furthermore, the MAWI working group is required to facilitate the exchange of measurement and analytic information across all working groups since all studies require some sort of measurement [56].

MAWILab is a database that provides academics with assistance in evaluating the effectiveness of their approaches for detecting traffic anomalies. It is made up of a series of labels that can identify traffic irregularities inside the MAWI archive. The labels are derived from the data by employing a sophisticated graph-based process that evaluates and integrates a number of separate and distinct anomaly detectors. The data set is updated every day to incorporate the latest traffic from forthcoming applications as well as any anomalies that may have occurred. Commodity traffic and research experiment traffic coexist in the WIDE backbone as their respective types of traffic. The majority of Japanese universities get their upstream via SINET (AS2907), and all of the universities that are members of WIDE also have multi-homed connections to SINET as well as WIDE. Therefore, not all of the traffic coming from these universities is carried by WIDE. In terms of international connectivity, WIDE is linked to Internet2 by way of TransPAC, and as a result, traffic to and from academic sites in the United States, the European Union, and Asia-Pacific travels along this route. Different sampling points are used to accumulate traffic traces from the MAWI traffic archive. We used traces from sample point F for our thesis.

3.5 Implementation

This section presents our python tool created to scrutinize the trace files derived from Internet Measurement. Different sections will explain the process of parsing a trace file and elucidate how each component of the system is being implemented. Sub section 3.5.1 will illustrate the json file and the fields essential for smooth functioning. Furthermore, sub section 3.5.2 will elaborate on the main source code and its performance.

3.5.1 Json Configuration

The JSON script is the initial and fundamental need for executing the program. Pertaining to the header fields of interest, the means by which we will retrieve their values, or the desired end result, the JSON script must

not contain any ambiguity for any discernible reason. A fragment of the system's required JSON file is below for reference. Detailed descriptions of the functionality of each field follow the code snippet.

```
1 {"proto":{
2   "ipv4":{
3     "and":[
4       {
5         "offset":0,
6         "length":1,
7         "value":5,
8         "bitmask":15,
9         "equal":"False",
10        "fn": "ihl"
11      },
12      {
13        "offset":9,
14        "length":1,
15        "value":[1,6,17],
16        "equal":"False",
17        "fn": "protocol"
18      }
19    ],
20
21    "ipv6":{
22      "and":[
23        {
24          "offset":1,
25          "length":3,
26          "value":0,
27          "ls":4,
28          "rs2":4,
29          "bitmask":1048575,
30          "equal":"False",
31          "fn": "flow_label"
32        }
33      ]
34    }
35 }
```

Listing 3.1: JSON Script

The field *"proto"*, will remain unchanged.

Succeeding it is the name of the protocol, the header information of which will be examined. The focus of this thesis will be on five protocol headers: IPv4, IPv6, TCP, UDP, and ICMP.

Function and name of the fields:

"name of the protocol": It depicts the protocol being used. There should be the following formats for the field names: ipv4, ipv6, tcp, udp and icmp

"and": This is an array-type JSON because, for various protocols, we needed to inspect multiple header fields. (Field name can't be changed.)

"offset": This field defines the header field's offset number. Its value can be in numbers only.

"length": It represents the length of the header field in bytes. It can have numerical value only.

"value": This field contains the value that needs to be found or the value that is to be excluded. Besides this, if we need to compare several values, the "value" field can alternatively take the form of an array.

"rs": "Right Shift"; This field is essential as the value for a header field is not always straightforward. For example, there can be two or more header fields in the same offset number. So, Bit-shifting is needed to get the correct packet's value for some header fields. The program is written in a manner that right/left shift needs to be done before the bitwise operation. So first, we will do all the bit-shifting and then the bitwise operation.

"ls": This field specifies the left shift.

"rs2", *"ls2"*: This field specifies the left shift. For some header fields, we need to do bit-shifting more than once. In the program, the order of doing the bit-shifting operation is RS - LS - RS2 - LS2. A slight deviation from the established order is possible but must be avoided. For example, suppose a field needs two right shifts or two rights and one left shift; then the order for the above situation will be RS-RS2 or RS-RS2-LS2. If no bit-shifting is required, one can skip the whole section.

"bitmask": Simply put, a 'bitmask' is a binary representation of data. It is what we get after doing AND operation with the value we get from the packet.

"equal": It can only be either true or false. If the resulting value needs to be the same as "value" in the JSON script, then equal will be "True"; otherwise, "False."

"fn": It represents a name for the header field.

3.5.2 Implementation Details

This section presents our design and implementation of the python tool. Figure 3.1 shows the flow of our code. The program is initiated by reading the trace file in its entirety and the json file. Then, using a function from python's `plt_module`, we start the trace file for analyzing each packet. Following this, an object is created specifically for the protocol being used by the packet and sent to its designated function for further processing. Once the object is inside its intended function, it creates a key to distinguish the packet from the rest of the packets and a json list from the json file. A copy of the object, json list, and an empty dictionary is passed to the main function. A copy of the object, json list, and an empty dictionary is passed to the main function.

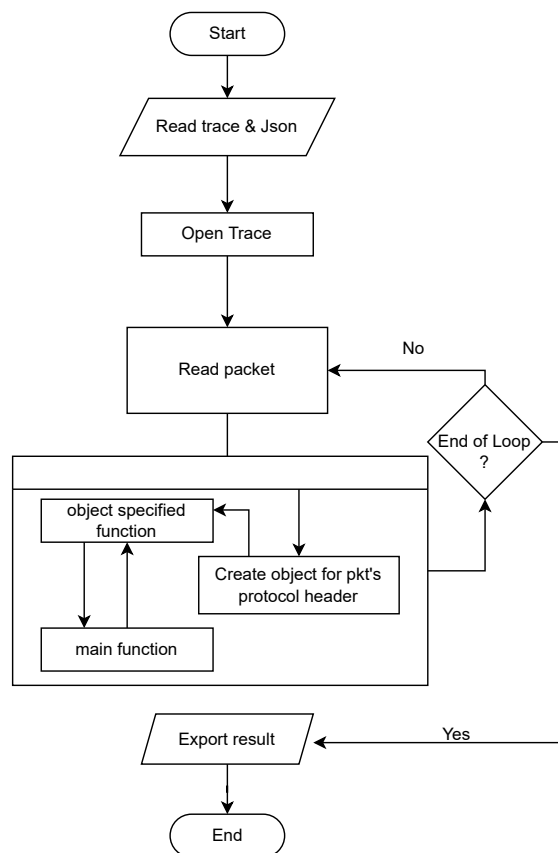


Figure 3.1: Flow Chart of the Tool

The main function is the heart of our tool. Inside the function, using the json configuration, we could look further into the packet's header fields to search for anomalies. If the specified header field value matches the json description, we save the value in the dictionary. When all the header fields defined in the json script are done evaluating, the function returns the dictionary storing all the unusual values. The unusual values are stored in a new dictionary with the key created before.

The process mentioned above continues for each packet until the loop ends.

After the loop is finished, we export the result and end the program. To better grasp how all the functions work, we will break down the source code into many code snippets and explain the tool more thoroughly with an example.

Assume we commenced the program by reading a pcap file and passing the json script shown in listing 3.1. We initialize two new pcap files named "sniffed" pcap to capture all the unusual values and "interesting_ip" to capture all the IP packets whose version is not 4 or 6, as "plt_module" can only handle IP packets with versions 4 and 6. Suppose the first packet we have is an IPv4 packet. In accordance to the listing 3.2, an `ip` object for the IPv4 packet's header is created and passed onto `ipv4function`.

```
1 startTime = time.time() #Calculate the time of the trace
   parsing
2 ot = plt.output_trace(f'pcapfile:sniffed_{str(sys.argv[1])}.
   rsplit('.', ) [0]+'.pcap') #store all the connections with
   interesting header field
3 ot_ip = plt.output_trace(f'pcapfile:interesting_ip_{str(sys.
   argv[1]).rsplit('.', ) [0]+'.pcap') #ip connctitions whose
   version is not 4/6
4 ot_ip.start_output()
5 ot.start_output()
6
7 for pkt in t:
8     ip = pkt.ip #to get the ipv4 object
9     ip6 = pkt.ip6 #to get the ipv6 object
10    tcp = pkt.tcp #to get the tcp object
11    udp = pkt.udp #to get the udp object
12    icmp = pkt.icmp #to get the icmp object
13
14    if ip:
15        ipv4function()
16        if udp:
17            udpfunction()
18    if ip6:
19        ipv6function()
20    if tcp: #for both ipv4 and v6
21        tcpfunction()
22    if icmp:
23        icmpfunction()
```

Listing 3.2: Read Packets

Since there are five distinct categories of objects, we have also defined five distinct functions. When the `ipv4function` object reach `ipv4function` a unique `ip_pair` key is generated with the packet's source address, destination address, and IP identification number. Despite the similarity between IPv4, IPv6, TCP, UDP, and ICMP in terms of their functions, the `ip_pair` format differs slightly between the five. It is depicted in listing 3.3 that as we are storing our data in dictionaries and the `ip_pair` is serving as a key.

An `ipv4_list` is also generated from the JSON script because this packet

utilize the IPv4 protocol. This list will be used to parse the JSON data. We also declare a variable to count the total IPv4 packets in the trace file. Then the list and empty dictionary are passed to the mainfunction.

```

1 def ipv4function():
2     try:
3         ip_pair = (str(ip.src_prefix), str(ip.dst_prefix), ip.
4             ident)
5         ipv4_list = jsonData['proto']['ipv4']['and']
6         global ipv4_pkt_count
7         ipv4_pkt_count += 1
8         mainfunction(ipv4_list, test_dict, ip)
9         if test_dict:
10            #print(test_dict)
11            ipv4_output[ip_pair] = dict(test_dict)
12            ot.write_packet(pkt)
13            test_dict.clear()
14 except:
15     ot_ip.write_packet(pkt)

```

Listing 3.3: IPv4 function

The piece of code in the listing 3.4 is critical. Our main task to look for anomalies in the header fields start here. We start by parsing the `test_list`. The first index of the list has all the important information to look for strange value in the internet header length (ihl) field. We used `offset` and `length` values from our JSON array to get the binary data from header's zero offset. IHL and version field share the same offset in IPv4 packet's header but to access the ihl in the header is pretty straightforward and we do not need any bit shift operation for this header field. So after the "AND operation" we accurately get packet's internet header length. After further inspection, if this is not the value we want, the packet will be ignored, and the entry will be recorded in the `test_dict` if it is the value of our interest.

```

1 def mainfunction(test_list, test_dict, pckt):
2     #print(pckt)
3     for i in range(len(test_list)):
4         equal = "False"
5         val_list = test_list[i]['value']
6         #list_check = isinstance(val_list, list)
7         length = test_list[i]['offset'] + test_list[i]['length',
8             ]
9         data_buffer = int.from_bytes(pckt.data[test_list[i]['
10            offset']:length], byteorder='big')
11         if "rs" in test_list[i]:
12             data_buffer = data_buffer >> test_list[i]['rs']
13         if "ls" in test_list[i]:
14             data_buffer = data_buffer << test_list[i]['ls']
15         if "rs2" in test_list[i]:
16             data_buffer = data_buffer >> test_list[i]['rs2']
17         if "ls2" in test_list[i]:
18             data_buffer = data_buffer << test_list[i]['ls2']
19         value = data_buffer & test_list[i]['bitmask']

```



```

18     if isinstance(val_list, list) == True:
19         for j in range(len(val_list)):
20             if value == val_list[j]:
21                 equal = "True"
22     else:
23         if value == test_list[i]['value']:
24             equal = "True"
25     if equal == test_list[i]['equal']:
26         test_dict.update({test_list[i]['fn'] : value})
27     return test_dict

```

Listing 3.4: Main Function

Now the packet repeat the same procedure for the next header field "protocol". When the packet is done checking all the header fields mentioned in the json file, the `test_dict` is sent back to the `ipv4function`. At that point, first we evaluate to see if the `test_dict` is empty. If the `test_dict` dictionary is not empty, we store that in a separate dictionary called `ipv4_output`. As for the other four protocols, we also have dictionaries for them, such as `tcp_output`, `udp_output` and so on and so forth. In addition to this, the packet is written into the sniffed trace file. Finally, the `test_dict` is clear out completely before proceeding on to the next packet.

The procedures and operations described above are similar to all `ipv6function`, `tcpfunction`, and `icmpfunction` functions.

All the result produced by this tool entirely depend on the JSON script. SO we may alter the output by just modifying the JSON script. However, we needed some tool updates for UDP headers.

For UDP, we are checking if the UDP length is equal to the total IP packet length minus IP header length, if yes then we check for checksum value otherwise store the length `udp_output`. We decided to make use of the python-libtrace (plt) module in order to directly access length and checksum fields through `udp.len` and `udp.checksum` from the UDP header, as shown in figure 3.5.

```

1 def udpfunction():
2     ip_pair = (str(udp.src_prefix), str(udp.dst_prefix), udp.
3         src_port, udp.dst_port)
4         udp_length = ip.pkt_len - ip.hdr_len - 15
5
6     if udp.len != udp_length:
7         udp_output[ip_pair] = {"Length": udp.len}
8         ot.write_packet(pkt)
9
10    else:
11        if (udp.checksum != 0 or udp.checksum == 0):
12            udp_output[ip_pair] = {"checksum": udp.checksum}

```

```
12 ot.write_packet(pkt)
```

Listing 3.5: UDP Function

We will begin by generating a pandas dataframe from the dictionaries, as illustrated in listing 3.6, and then move on to generating the final findings.

```
1 ipv4 = pd.DataFrame.from_dict({i: ipv4_output[i]
2                               for i in ipv4_output.keys()},
3                               orient='index')
4 ipv4.to_csv(r'output_ipv4_'+str(sys.argv[1]).rsplit('.', 1)[0]+'
5            .csv')
```

Listing 3.6: Generate output

3.5.3 Result Code

In this section, the observations from our research will be analyzed. We need to organize the data in a meaningful way. We select pandas for our data analysis and visualization. Panda's python library includes data structures designed to speed up, generalize, and expressly work with "relational" or "labeled" data. This is a fundamental building block for conducting practical, everyday data analysis in Python. The data types that pandas can process are pretty extensive. In addition, custom packages built atop pandas are used more frequently to address data preparation, analysis, and visualization. After we get all the output data files from the main program, we process them in a substantial manner. The analysis of the results is conducted in two stages, i.e., the results are initially broken down by month and subsequently by the entire year.

Since we have parsed daily data from June to December in 2020, January til March in 2021 as well as in 2022, we can say that we have a good grasp of the situation. The information for each day of the month is included in the output generated by the primary program. In addition, the number of days varies between months, not only due to the actual number of days per month but also the fact that not every one of these days has an out-of-the-ordinary value in the header field.

The code snippet 3.7 is an example of how we process IPv6 outputs. For IPv6 protocol we are specifically interested in "flow label", "traffic class" and "next header" fields. A detailed description of the interesting header fields we examine in this thesis and why they were selected is given in Section 4.1.

First, we check if the column exists in the dataframe. As not all days have strange values, so output files for same protocol may not have same columns. Then we read the column values from the dataframe. For the "flow label" and "traffic class" columns, we are only interested when the values are not zero, so counting all the non-zero values and putting them in a new column name `count_non_zero`. Moreover, we want all the distinct

values for the "next header" column. `value_counts()` function counts how many times each unique value appears for that day. `to_frame().T` is used to interchange the rows and columns, so the unique values become columns and their value count become row data. To make the data process more manageable in the future, we insert three new columns named `year`, `month`, and `date`.

Repeating the same manner, we process the output files for other protocols.

```

1     try:
2         df = pd.read_csv(filename)
3
4         if 'flow_lable' in df.columns:
5             column = df['flow_lable']
6             count_non_zeros = column[column != 0].count()
7             result_flow_lable.loc[len(result_flow_lable), [
'year', 'month', 'date', 'count_non_zero']] = year, month,
date, count_non_zeros
8
9         if 'traffic_class' in df.columns:
10            column = df['traffic_class']
11            count_non_default = column[column != 0].count()
12            result_traffic_class.loc[len(
result_traffic_class), ['year', 'month', 'date', '
count_non_default']] = year, month, date, count_non_default
13
14            if 'next_hdr' in df.columns:
15                data_next_hdr = df['next_hdr'].value_counts(
dropna=False)
16                data_next_hdr = data_next_hdr.to_frame().T
17                result_next_hdr = pd.concat([result_next_hdr,
data_next_hdr], ignore_index=True)
18                result_next_hdr.at[b, 'date'] = date
19                b += 1
20            except:
21                pass

```

Listing 3.7: Generate Result

Figure 3.8 depicts an essential phenomenon. The total packet counts from the `output_packet_count.csv` file are essential to calculate the total default value before processing the final result. For example, suppose `df1` is the dataframe for the "packet count" file for IPv6 and `df2` is the dataframe of the "result" file we obtained from the last code. Then, we define the same index for both dataframes. However, the index count is not exact for both dataframes because we counted the total number of packets for all days regardless of whether that day had any strange values. So, by matching the indexes (`year`, `month`, `date`), we calculate the default values and join `total` and `default_val` in the previous result file.

```

1 #define columns as index
2 df1 = df1.set_index(['year', 'month', 'date'])
3 df2 = df2.set_index(['year', 'month', 'date'])
4 #list of the index

```

```

5 i1 = df1.index
6 i2 = df2.index
7
8 val_sum = df2[col_list].sum(axis=1)
9 default_value = df1.loc[i2][protocol] - val_sum
10 total = df1.loc[i2][protocol]
11
12 result = df2.assign(default_val=np.where(i2.isin(i1),
13     default_value , 0),
14     total=np.where(i2.isin(i1), total , 0))
15 #reset the multi index
16 final_df = result.reset_index()
17 #exporting the csv
18 final_df.to_csv(f'final_{year}_{protocol}_{field}.csv', index =
    False)

```

Listing 3.8: Join Files

After all the processing done so far, we have created a separate CSV file for each header field for a given year. This will come in handy if we need to inspect results more thoroughly for a specific year. In the final stage of our result finding, which is depicted in listing 3.9, we will first take into consideration all of the days that are contained within a month, and then we will divide the total number of packets for the targeted protocol by the total number of items contained within each column.

```

1 for year in years:
2     fn2 = f'final_{year}_{protocol}_{field}.csv'
3     df2 = pd.read_csv(fn2)
4     col_list= list(df2)
5     col_list = [e for e in col_list if e not in ('year', 'month',
6         'date')]
7
8     df2 = df2.fillna(0)
9     df = df2.groupby(['year', 'month'])[col_list].apply(lambda
10     x : x.astype(int).sum()).reset_index()
11     month_df = pd.concat([month_df, df],ignore_index=True)
12
13 month_df = month_df.fillna(0)
14 col_list1= list(month_df)
15 col_list1 = [e for e in col_list1 if e not in ('year', 'month',
16     'total')]
17
18 percentage_df = month_df.copy()
19 for col in col_list1:
20     percentage_df[col]=month_df[col]/month_df['total']*100

```

Listing 3.9: Final Result

Chapter 4

Result

This section describes the findings from the overall analysis done in this thesis.

4.1 Interesting Header Fields

Usually in the analysis of the Header Fields, 'anomalies' probably mean that "a new functionality is used", but in this thesis, an "anomaly" is defined as a relatively rare or unusual event, instead of the occurrence of an extremely rare event. After some deliberation, we settled on a set of header fields to examine in this thesis, all with the aim of discovering whether any seldom used protocol mechanisms have evolved over time.

Following are the crucial results inferred from the research and categorized under respective protocols:

Subsequent are the interesting header fields for IPv4:

"IHL": This option sets a fixed header length because IPv4 header sizes can fluctuate. The length of IPv4 headers might vary. The header grows proportionally to the number of configurable parameters. An option instructs intermediary devices on how to forward or handle data packets. A value other than 5 will indicate the use of IPv4 options.

"DSCP": This 6-bit field indicating a packet's network quality can store and retrieve values from 0 to 63. Values other than 0 are of interest to us and represent the extent to which it has evolved.

"ECN": Notification of congestion can be sent via ECN, without packet loss. ECN-capable Transport (ECT) and Congestion Experienced (CE) bits make up an ECN-specific IP header field. These two bytes match IP header DSCP bits 6 and 7. If the value is not 0, it is valuable for our analysis.

"Protocol": We find that the protocols Transmission Control Protocol(TCP), User Datagram Protocol (UDP), Internet Message Access Protocol version 4 (ICMPv4), Generic Route Encapsulation (GRE) and IPv6

Encapsulation (IPv6) are the most typical; we speculate if any out-of-the-ordinary protocols have become increasingly popular or are in use.

Header fields to be investigated for IPv6:

"Flow Label": A source can utilize the Flow Label field to mark packets that need non-standard quality of service (QoS) or real-time support from intermediary IPv6 routers. Multiple flows may exist between a source and a destination due to simultaneous activities. Besides this, default routers and hosts that don't implement the flow label field leave it at 0. If the value is not equal to 0, it's useful for our analysis.

"Traffic Class": This field is the IPv6 equivalent of 'DSCP'.

"Next Header": With some additional options, the field is similar to the IPv4 Protocol field. So, it is interesting to see what protocols are used most other than TCP, UDP, ICMP and Fragment header (44).

For TCP, we are interested in the following header fields:

"RSV": The reserved field uses three digits. They are never utilized and always have a value of 0. If it's not 0, further analysis is done.

"CWR, ECE": Not defined in the initial TCP specification, ECE and CWR were once known as XMAS or YMAS. ECE is used to notify senders of network congestion, preventing packet loss and subsequent re-transmissions. The sender uses CWR to indicate reception of ECE = 1. If it is not 0 then it is of our interest.

The header field we are interested in for UDP is:

"Length field": The field is ignored if the length is equal to the total packet's length minus the IP header length. But if that's not the case then it's interesting, possibly meaning a UDP option is being used. UDP option is a somewhat new concept, so we wanted to see if the use of this field has changed over time.

We are interested in the following header field for ICMP:

"Type": The ICMP packet type is specified in the type field. A type 3 is assigned to ICMP Destination Unreachable packets, for instance. This variable length (in bytes) field has a total of 8. The usage of these values and the variants in them are of interest.

4.2 Result Analysis

This section will explain the ultimate outcome for all the header fields mentioned in 4.1. The results deduced from the regular analysis of the selected significant fields are explained below.

IHL: We have looked at every possible value except five that might interest us. And from the analysis we found the percentage of our interest values or non default values of the IPV4's total packets range from 0.000 to 0.014 and there is no clear trend. Despite the negligible deviation from the default value, the fact that IPv4 options are being used in any way is indicated by this small percentage.

ECN : Figure 4.1 shows that CE and ECT(1) are consistently near zero, while the increasing prevalence of ECT (0) is intriguing. The usage of ECT(0) remained relatively stable between 0.75% and 1.25% during 2020 and 2021, before increasing in the first two months of 2022, only to fall back below 1% by the third month of that year. Despite the lack of a consistent trend, we can reasonably assert that ECT(0) is employed more frequently than CE and ECT (1).

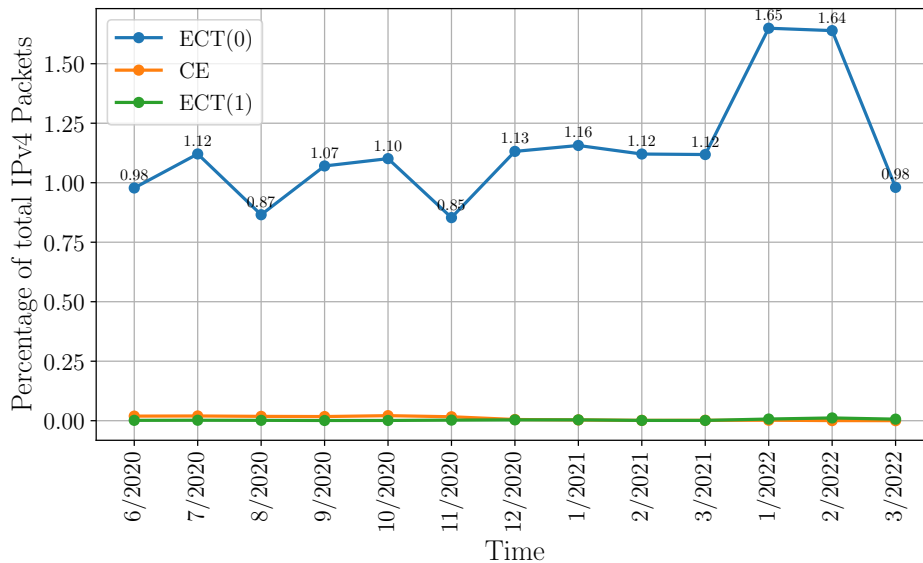


Figure 4.1: IPv4 ECN values

DSCP: The percentage of non default or interesting values range from 0.34 to 1.04. From June, 2020 until December, 2020 the values always remain between 0.4 and 0.6 with an increase in November 2020 to 1.04 percent. Then in 2021 the values drop below 0.4 percent and increased above 0.6 percent in 2022. There is no discernible pattern.

Protocol: IPsec consists of the Authentication Header (51) and the Encapsulating Security Payload (50). For this reason, we have combined ESP and AH as IPsec in the graph. IPsec’s value went from 0.04 percent to 0.11 percent between June 2020 and March 2021. In 2022, however, it fell below 0.03 percent and appeared to continue falling. Though an increase in Stream Control Transmission Protocol (SCTP) usage would have been intriguing, the value consistently trended downwards. The trend of IPsec and SCTP is depicted in Figure 4.2.

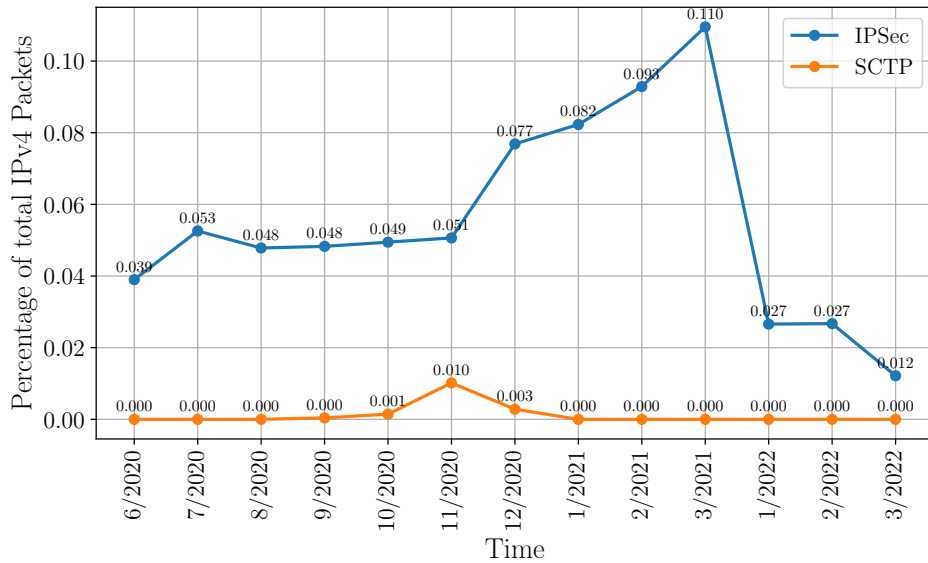


Figure 4.2: IPv4 Protocol

Flow Label: Figure 4.3 depicts the percentage of the non-default value, defined as anything other than zero for the flow label, fluctuating from 1.4% to 2.2% in the year 2020. After staying flat in 2021, the graph dropped to 1.07% in February 2022 before beginning an upward trend once more in March of that year. Seeing the graph we assume that this occurs either because of the use of quality of service (QoS) or because the flow’s source node assigns a flow label.

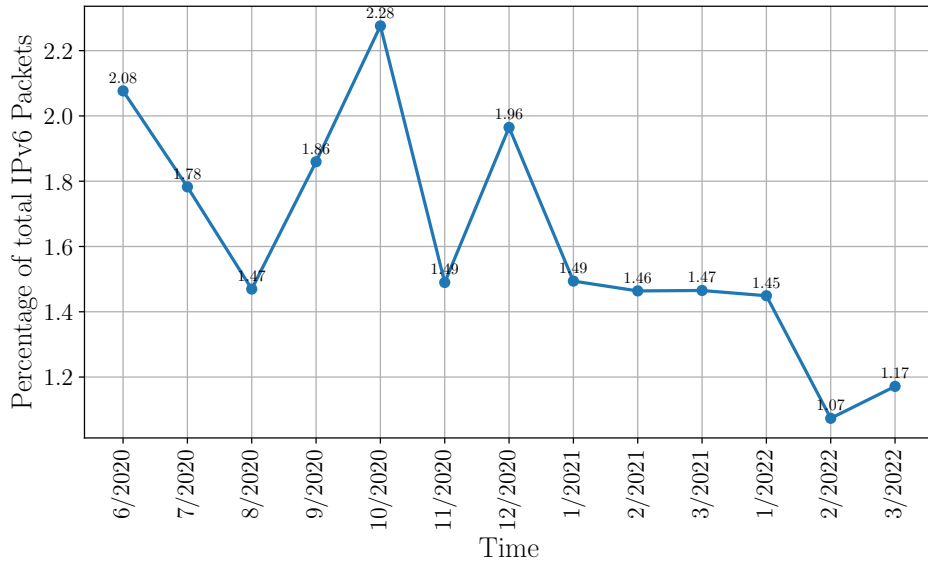


Figure 4.3: IPv6 flow label

Traffic Class: Traffic class analysis yields no significant results. Between 0.02 percent and 0.10 percent of all IPv6 packets had traffic class information. However, the result showed a continuous decline throughout the years.

Next Header: Table 4.1 shows the cumulative occurrences of all the protocols listed there. While these numbers are tiny compared to the overall quantity of IPv6 packets, they do show the consistent use of Protocol Independent Multicast (PIM) with IPv6 over time, which is noteworthy. PIM, or Protocol Independent Multicast [57], is a multicast forwarding protocol suite for specific use cases. As opposed to including its own topology identification process, PIM relies on routing parameters provided by other protocols used in the routing process. It facilitates multicast routing without requiring the use of specialized unicast routing techniques. It also constructed both source and shared distribution trees, which are used to forward packets from numerous sources. Furthermore, each network group address is represented in PIM as a node in the shortest path tree.

TCP ECE & CWR: Assuming an ECN-aware network, routers, for example, will set the IP header's CE flag when they encounter high data volumes that could lead to congestion or dropped packets. The receiver informs the sender of such a CE-mark using the ECE flag. This may trigger a TCP Slow Start. Upon receiving a TCP segment with the ECE flag set, the sender sets the CWR flag, halving the send window and reducing the slow start threshold. Nevertheless, both ECE and CWR are rarely seen in connections. The graph indicates correct use of ECE and CWR among very few hosts who are using it.

Year	Month	Total IPv6 Packets	IPv6 hop-by-hop option (0)	Destination Options for IPv6 (60)	Protocol Independent Multicast (103)
2020	6	142.67M	0	0	290
2020	7	104.28M	1	2	299
2020	8	93.30M	2	0	298
2020	9	63.93M	9	0	269
2020	10	77.83M	2	0	261
2020	11	93.84M	8	0	260
2020	12	86.42M	4	0	267
2021	1	92.63M	1	0	242
2021	2	88.68M	0	0	243
2021	3	98.14M	0	0	289
2022	1	66.17M	3	1	213
2022	2	81.61M	0	22	189
2022	3	67.84M	0	12	178

Table 4.1: IPv6 next header field

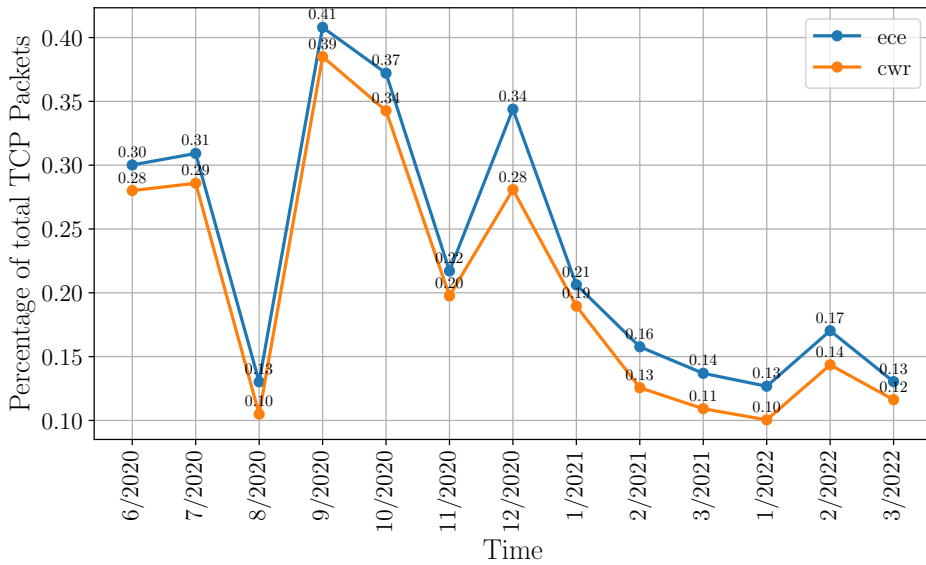


Figure 4.4: TCP ECE & CWR Field

Reserved: Typically the RSV field is always set to 0. We have plotted two y-axis in the graph shown in figure 4.5. Left side shows the percentage of the total TCP packets and right side is the total Number of packets have set RSV field not zero. There is no visible trend but from December 2020 til January 2021 there was a rise for all the RSV field values. It is also visible in the graph that RSV 1 is always higher than other RSV values except January 2021 and March 2022.

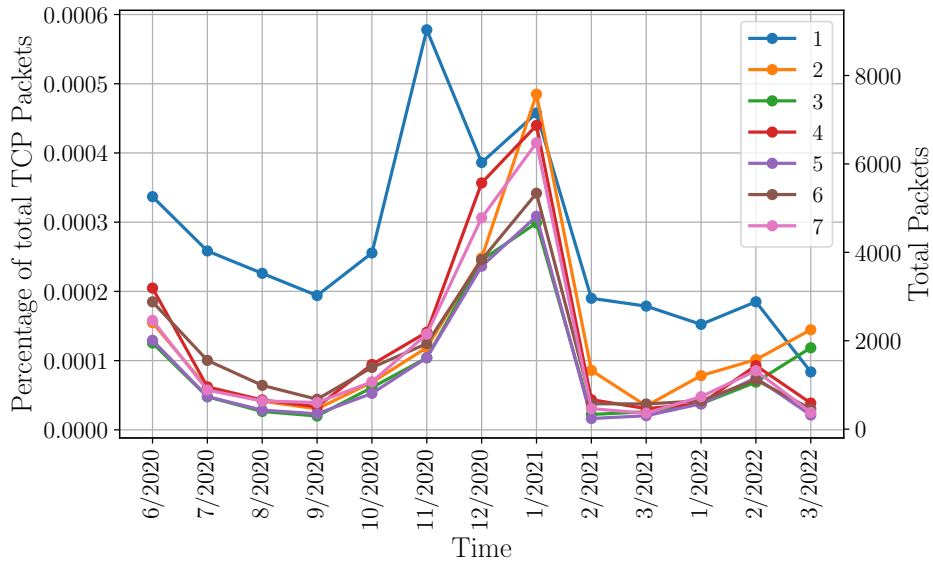


Figure 4.5: TCP Reserved Field

Length/Option: There is no header option space available for UDP. An option extends the capabilities of a transport protocol. Besides this, a UDP header provides only ports and a checksum to detect errors, adding minimal functionality to the IP. A trailer area is included after the UDP user data in an IETF draft [58] to enable such options. The results of our research show that the length field percentage might be anything from 0.01 to 0.13 percent. This option field is still an experiment work, so it is fascinating to see the use of a UDP option.

Type: Findings for ICMP type header field is quite interesting. Source Quench messages are used to control congestion. But this type 4 source quench messages are deprecated [59]. But from the table we see the use of this type 4 throughout 2020. When a host encounters difficulty parsing a packet's header information, it will issue a message of type 12. This is exactly what we were on the lookout for in this thesis; thus, it will be really illuminating to take a look at those packets. In addition, the widespread adoption of Network Time Protocol (NTP) raises the question of why the concepts of the Timestamp and the Timestamp reply are still in use.

Year	Month	Total	Source Quench (4)	Unassigned (1)	Parameter Problem (12)	Timestamp and Timestamp Reply (13+14)
2020	6	1000.41M	376	1	35	599
2020	7	1034.22M	356	0	61	13746
2020	8	1051.28M	424	1	52	4005
2020	9	946.09M	430	0	25	1035
2020	10	975.17M	307	0	30	1198
2020	11	977.19M	314	1	43	1181
2020	12	122.91M	36	2	78	1259
2021	1	14.25M	0	0	3	173
2021	2	14.89M	3	0	10	62
2021	3	16.39M	0	6	15	101
2022	1	35.53M	0	339	38	76
2022	2	32.00M	1	317	27	57
2022	3	29.64M	0	254	4	152

Table 4.2: ICMP Type

Chapter 5

Discussion and Conclusion

Different conclusions are drawn based on what has been discussed and analyzed in this thesis, and potential prospects for future work are proposed.

5.1 Result Findings

At the very first chapter, in section 1.4, there was a research question that this thesis will address.

Research Question: How much have packet header fields changed over time?

Answer: There are no discernible patterns suggesting a likely shift in protocol mechanism. Also, no distinct changes in header fields are visible after analyzing the data.

However, some of the discoveries in Chapter 4 are noteworthy, as presented by the result.

- A great scope lies related to IPv6's next header field and PIM. The default setting in PIM triggers a message to be sent every 60 seconds. Our finding showed that there were always some PIM values with IPv6. First of all, it is unusual for PIM to use more in IPv6 than IPv4. And secondly, someone is out there using PIM.
- The result from the ICMP type field is quite interesting. We found usage of forbidden source quench messages; some packets have anomalies in their header field and the use of Timestamp and Timestamp reply. It is possible to look deeper into these packets to determine the exact timing and what happened at that time with what environment configuration they had.
- Even though RSV findings are insignificant compared to the total number of packets. However, from our findings, we noticed that during a specific period, there was a sudden rise for all RSV field values, which indicates either a measurement test campaign or someone is experimenting with them.

- Findings for IPv4's IHL, DSCP and protocol fields and IPv6's traffic class is not worthy of more inspection. However, IPv4's ECN, TCP's ECE & CWR, IPv6's flow label, and UDP's option fields can be an interest of research for the future.
- While parsing data for 2021, we got some IP packets with version fields other than 4 and 6. After inspecting the "interesting_ip" pcap file, we found Bogus packets with version 7.

5.2 Limitations

While these results are promising, they should be interpreted cautiously, and some limitations should be kept in mind.

- This thesis relies on MAWI trace files, yet each day's worth of data only amounts to fifteen minutes' of traces. It is essential to have a sufficient sample size in order to draw valid conclusions, which our thesis lacks. A larger dataset may improve the outcomes.
- Our dataset was scattered. A continuous dataset might have helped us get a more precise result.
- There are limited prior research works related to this topic. Even though we have lots of traffic analyzing tools available, only a few studies have been done to detect anomalies in today's internet traffic.
- The functionality of the software is also restricted. When processing bigger pcap files, the software may become unresponsive or even crash. We faced this mainly because of hardware problems. Using a better and more powerful machine while working with trace files is recommended. It would be feasible to obtain an update on this tool's version.

5.3 Future work


This research on the packet header fields was carried out by us using traffic captures from the MAWI dataset. Therefore, one can use different datasets, such as CAIDA, to compare the findings for future work. On the other hand, developing a traffic analysis tool is challenging due to the slight differences between each header format. However, more analysis tools could be developed for the research applications using the tool developed and portrayed in this thesis. Furthermore, advanced visualization and graph-making techniques could be used by adding more features to the developed tool.

This thesis investigates the use of packet header fields in trace files. The outcome of the thesis result may be insignificant due to insufficient sample size, but that leaves considerable room for improvement. However, the result shows that by making changes to the Json file, one can gain access to the

offsets and perform a more in-depth analysis of the packet. Also, our tool provides us with several sniffed pcap files containing all anomalous packets, allowing us to investigate them further. In light of these arguments, it can be established that more research can be done to develop this thesis fully.

Appendix A

Source Code

The source code for the program can be found here: <https://github.com/naimans/packet-trace> / 

Files needed for the program:

- plt_testing.py
- main.py
- json_data.json

To make use of the tool, one will need to execute the shell command that is provided below:

```
1 sudo python3 main.py X.pcap json_data.json
```

Bibliography

- [1] Cong Tang et al. 'A general traffic flow prediction approach based on spatial-temporal graph attention'. In: *IEEE Access* 8 (2020), pp. 153731–153741.
- [2] Zhibin Zuo et al. 'P4Label: packet forwarding control mechanism based on P4 for software-defined networking'. In: *Journal of Ambient Intelligence and Humanized Computing* (2020), pp. 1–14.
- [3] *Ossification of the Internet*. <https://www.scs.stanford.edu/nyu/04sp/notes/l23.pdf>. (Accessed on 11/15/2022).
- [4] Anurag Kumar, D Manjunath and Joy Kuri. *Wireless networking*. Elsevier, 2008.
- [5] James D McCabe. *Network analysis, architecture, and design*. Elsevier, 2010.
- [6] *What is the OSI Model? 7 Network Layers Explained | Fortinet*. <https://www.fortinet.com/resources/cyberglossary/osi-model>. (Accessed on 10/26/2022).
- [7] Paul Simoneau. 'The OSI Model: understanding the seven layers of computer networks'. In: *Expert Reference Series of White Papers, Global Knowledge* (2006).
- [8] *TCP/IP Model - GeeksforGeeks*. <https://www.geeksforgeeks.org/tcp-ip-model/>. (Accessed on 09/24/2022).
- [9] James F Kurose and Keith W Ross. *Computer Networking, ; Boston, ua, 2003*.
- [10] *What are Network Packets and How Do They Work?* https://www.techtarget.com/searchnetworking/definition/packet?_gl=1%2A1xylwm%2A_ga%2AMTlxOTk2. (Accessed on 09/03/2022).
- [11] Ray Hunt. 'Transmission Control Protocol/Internet Protocol (TCP/IP)'. In: (2003).
- [12] K. Ramakrishnan, S. Floyd and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. <http://www.rfc-editor.org/rfc/rfc3168.txt>. RFC Editor, Sept. 2001. URL: <http://www.rfc-editor.org/rfc/rfc3168.txt>.
- [13] J. Touch. *Updated Specification of the IPv4 ID Field*. RFC 6864. <http://www.rfc-editor.org/rfc/rfc6864.txt>. RFC Editor, Feb. 2013. URL: <http://www.rfc-editor.org/rfc/rfc6864.txt>.

- [14] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. STD 86. RFC Editor, July 2017.
- [15] Kathleen Nichols et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474. <http://www.rfc-editor.org/rfc/rfc2474.txt>. RFC Editor, Dec. 1998. URL: <http://www.rfc-editor.org/rfc/rfc2474.txt>.
- [16] *Internet Protocol version 6 (IPv6) Header - GeeksforGeeks*. <https://www.geeksforgeeks.org/internet-protocol-version-6-ipv6-header/>. (Accessed on 10/21/2022).
- [17] J. Rajahalme et al. *IPv6 Flow Label Specification*. RFC 3697. RFC Editor, Mar. 2004.
- [18] D. Borman, S. Deering and R. Hinden. *IPv6 Jumbograms*. RFC 2675. RFC Editor, Aug. 1999.
- [19] *IPv6 packet*. https://en.wikipedia.org/wiki/IPv6_packet. (Accessed: 2021-04-10).
- [20] N. Spring, D. Wetherall and D. Ely. *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*. RFC 3540. RFC Editor, June 2003.
- [21] *TCP flags. There are six original 1-bit control...* | by CyberBruhArmy | LiveOnNetwork | Medium. <https://medium.com/liveonnetwork/tcp-flags-4e2df36c1a9d>. (Accessed on 11/27/2022).
- [22] Adam Dunkels Jean-Philippe Vasseur. *Interconnecting smart objects with ip: The next internet*. Morgan Kaufmann, 2010.
- [23] *Network Basics: TCP/IP Protocol Suite*. <https://www.dummies.com/article/technology/information-technology/networking/general-networking/network-basics-tcpip-protocol-suite-185407/>. (Accessed: 2021-08-05).
- [24] *Transmission Control Protocol (TCP) (article)*. (Accessed: 2021-04-15). URL: <https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d/the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp>.
- [25] J. Postel. *User Datagram Protocol*. STD 6. RFC Editor, Aug. 1980. URL: <https://datatracker.ietf.org/doc/html/rfc768>.
- [26] *Explain UDP with its header format?* <https://www.ques10.com/p/27217/explain-udp-with-its-header-format-1/>. (Accessed: 2022-02-17).
- [27] J. Postel. *Internet Control Message Protocol*. RFC 777. RFC Editor, Apr. 1981. URL: <https://datatracker.ietf.org/doc/html/rfc777>.
- [28] *What Is The Size Of Icmp Packet? [Comprehensive Answer]*. <https://answeregy.com/what/what-is-the-size-of-icmp-packet.php>. (Accessed: 2021-04-10).
- [29] A. Conta, S. Deering and M. Gupta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443. RFC Editor, Mar. 2006. URL: <https://datatracker.ietf.org/doc/html/rfc4443>.

- [30] Joy Kuri Anurag Kumar D. Manjunath. *Communication networking: an analytical approach*. Academic Press, 2004, pp. 235–243.
- [31] Shane Alcock, Perry Lorier and Richard Nelson. ‘Libtrace: A packet capture and analysis library’. In: *ACM SIGCOMM Computer Communication Review* 42.2 (2012), pp. 42–48. DOI: [10.1145/2185376.2185382](https://doi.org/10.1145/2185376.2185382).
- [32] Praful Saxena and Sandeep Kumar Sharma. ‘Analysis of network traffic by using packet sniffing tool: Wireshark’. In: *International Journal of Advance Research, Ideas and Innovations in Technology* 3.6 (2017), pp. 804–808.
- [33] N Patel, R Patel and D Patel. ‘Packet sniffing: network wiretapping’. In: *IEEE International Advance Computing Conference (IACC 2009) Patiala, India*. 2009, pp. 6–7.
- [34] *Packet Loss Explained - Causes and Best Solutions | IR*. <https://www.ir.com/guides/what-is-network-packet-loss#anchor1>. (Accessed on 08/04/2022).
- [35] Muhammad Sajid Mushtaq and Abdelhamid Mellouk. *Quality of experience paradigm in multimedia services: application to OTT video streaming and VoIP services*. Elsevier, 2017.
- [36] Ghislaine Livie Ngangom Tiemeni. ‘Performance estimation of wireless networks using traffic generation and monitoring on a mobile device.’ In: (2015).
- [37] Romain Thibault Fontugne et al. ‘Increasing reliability in network traffic anomaly detection’. PhD thesis. Citeseer, 2011.
- [38] Hong Wang et al. ‘Detection network anomalies based on packet and flow analysis’. In: *Seventh International Conference on Networking (ICN 2008)*. IEEE. 2008, pp. 497–502.
- [39] Evan Hughes and Anil Somayaji. ‘Towards Network Awareness.’ In: *LISA*. 2005, pp. 113–124.
- [40] Oded Shimon. *BruteShark*. <https://github.com/odedshimon/BruteShark>.
- [41] *Documentation - eCAP*. <https://www.e-cap.org/docs/>. (Accessed on 09/19/2022).
- [42] *About - DPDK*. <https://www.dpdk.org/about/>. (Accessed on 09/19/2022).
- [43] Eddie Kohler. *IPsumdump*. <https://github.com/kohler/ipsumdump>.
- [44] Shawn Ostermann. *tcptrace - Official Homepage*. <http://www.tcptrace.org/index.shtml>. (Accessed on 09/20/2022).
- [45] Simson L. Garfinkel. *tcpflow*. <https://github.com/simsong/tcpflow>.
- [46] *TCPDUMP & LIBPCAP*. <https://www.tcpdump.org/>. (Accessed on 09/20/2022).
- [47] Runa Barik et al. ‘fling: A flexible ping for middlebox measurements’. In: *2017 29th International Teletraffic Congress (ITC 29)*. Vol. 1. IEEE. 2017, pp. 134–142.

- [48] Gregory Detal et al. 'Revealing middlebox interference with Tracebox'. In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 1–8.
- [49] *Endace Measurement System Ltd.* <https://www.endace.com/>. (Accessed on 11/27/2022).
- [50] University of Waikato. *Libtrace - Centre for Open Software Innovation.* <https://cosi.cms.waikato.ac.nz/projects/wand-network-research/libtrace>. (Accessed on 10/05/2022).
- [51] *python-libtrace.* <https://www.cs.auckland.ac.nz/~nevil/python-libtrace/>. (Accessed on 10/05/2022).
- [52] *CAIDA Data - Completed Datasets - CAIDA.* <https://www.caida.org/catalog/datasets/completed-datasets/>. (Accessed on 05/13/2022).
- [53] *MAWI Working Group Traffic Archive.* <https://mawi.wide.ad.jp/mawi/>. (Accessed on 05/11/2022).
- [54] *WIDE - Idea.* https://www.wide.ad.jp/Vision/index_e.html. (Accessed on 05/13/2022).
- [55] *WIDE - Working Groups.* https://www.wide.ad.jp/Groups/index_e.html. (Accessed on 05/13/2022).
- [56] *WIDE MAWI WorkingGroup.* <http://mawi.wide.ad.jp/>. (Accessed on 05/13/2022).
- [57] D. Estrin et al. *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*. RFC 2362. RFC Editor, June 1998.
- [58] *draft-ietf-tsvwg-udp-options-18.* <https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-udp-options>. (Accessed on 11/18/2022).
- [59] F. Gont. *Deprecation of ICMP Source Quench Messages*. RFC 6633. <http://www.rfc-editor.org/rfc/rfc6633.txt>. RFC Editor, May 2012. URL: <http://www.rfc-editor.org/rfc/rfc6633.txt>.