Tampere University

Omolola Martha Shokunbi

COMPARISON AND FEASIBILITY OF NETWORK HETEROGENEITY AND INSPECTION TOOLS FOR QUIC

Masters of Science Thesis
Information Technology and Communication Sciences
Supervisor: Bilhanan Silverajan
October 2023

ABSTRACT

Omolola Martha Shokunbi: Comparison and Feasibility of Network Heterogeneity and Inspection Tools for QUIC
Masters of Science Thesis
Tampere University
Information Security
October 2023

QUIC protocol founded by Google and standardized by IETF, is a fairly new protocol that is here to stay. It has gained popularity among large organizations, which implies it acceptance. The weaknesses of HTTP – presumed to be the future of the web, which could not handle the high bandwidth and latency sensitive applications, brought about the development of QUIC. HTTP/3 is solely built on QUIC and makes the best use of QUIC's properties. This thesis aims at comparing QUIC under different network conditions, to see if the different properties of QUIC hold and how well they perform. In making this comparison, the QUIC traffic had to be inspected, and the tools used for this inspection were assessed based on their feasibility. To be able to understand the properties of QUIC, literature review was done. After which live experiments were done to see that the properties held. This thesis helped to understand the current state of QUIC, the security it provides to its users, and how feasible the tools used for carrying out inspection were. There were some limitations to what the tools could achieve, based on its intended usage. The visualization tool – QVIS, could not be used to illustrate the captured file from wireshark, which was a drawback. Samples from the qvis github were used in this paper, to demonstrate how the visualization would have looked.

Keywords: QUIC, HTTP, QLOG, QVIS

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

This thesis was a prerequisite in partial fulfillment of my Master of Science in Information Technology at Tampere University, Finland. This thesis was supervised by Bilhanan Silverajan, Tampere University's Postdoctoral Research Fellow and Hanning Zhao, Doctoral Researcher at Tampere University.

Firstly, I would like to thank God for the grace given to me, to be able to start my program and complete the program, with the submission of this thesis. It would not have been possible without his infinite mercies. Secondly, thanks goes to Bilhanan Silverajan for being a wonderful supervisor, all the times he took out from his busy schedules to meet with me, and his pleasant guidance. I am grateful to you for believing in me to deliver this thesis. I would like to thank my parents, Mr and Mrs Shokunbi for their unwavering support throughout my thesis writing and my program as a whole. And to my aunt and her husband, Pastor and Pastor Mrs Sunmonu, they provided support both materially and spiritually, which I appreciate immensely. My brothers who constantly harassed me about completing my thesis, I am done now, and I appreciate their own way of showing support. My appreciation would not be complete without thanking you, Ope, you stood by me all through this, and I thank you :*.

Thank you to all my family and friends, it truly does take a community to raise a child.

Tampereella, 17th October 2023

Omolola Martha Shokunbi

CONTENTS

1. Introduction . . . 1
- 1.1 Overview . . . 1
- 1.2 Research Questions . . . 2
- 1.3 Scope . . . 2
- 1.4 Methodology . . . 2
- 1.5 Structure of the Thesis . . . 3

2. Background . . . 4
- 2.1 HTTP . . . 4
- 2.2 Why QUIC? . . . 5
- 2.3 What is QUIC? . . . 5
- 2.4 Properties of QUIC . . . 8
  - 2.4.1 Security . . . 8
  - 2.4.2 Stream Multiplexing . . . 10
  - 2.4.3 Lower Latency . . . 11
  - 2.4.4 Congestion Control . . . 13
  - 2.4.5 Flow Control . . . 14
  - 2.4.6 Forward Error Correction . . . 14
  - 2.4.7 Connection Migration . . . 14
- 2.5 QLOG and QVIS . . . 15
  - 2.5.1 QLOG . . . 16
  - 2.5.2 QVIS . . . 17
  - 2.5.3 Sequence Tool . . . 17
  - 2.5.4 Packetization Tool . . . 17
  - 2.5.5 Congestion Tool . . . 17
  - 2.5.6 Multiplexing Tool . . . 18
- 2.6 Related Works . . . 18

3. Design and Implementation . . . 19
- 3.1 Design and Implementation . . . 19
- 3.2 Experimental Setup . . . 19
  - 3.2.1 Experiment Topology . . . 19
  - 3.2.2 Software Tools . . . 19
  - 3.2.3 QUIC Client and Server . . . 20
  - 3.2.4 Setting up the Connection . . . 21
  - 3.2.5 The Setups . . . 21

3.2.6 Setup A 21
3.2.7 Setup B 22
3.3 Traffic Conditions 24
4. Results and Analysis 27
4.1 Capture of the Different Traffic Conditions 27
4.1.1 Baseline 27
4.1.2 GEO Satellite 28
4.1.3 LTE 28
4.2 Visualization of Captured Traffic 28
4.2.1 Diagrammatic Representation of the Visualization Tools 29
5. Conclusion 39
References 41

LIST OF FIGURES

2.1 Architecture of QUIC Protocol [12] 6
2.2 Short Header [2] 6
2.3 Long Header [2] 7
2.4 QUIC Header [2] 7
2.5 Version Negotiation [25] 8
2.6 Header Structure of QUIC packet [26] 8
2.7 HTTP Request over TCP + TLS [11] 10
2.8 HTTP Request over QUIC [11] 11
2.9 Multiplexing [3] 12
2.10 QUIC Connection Establishment [2] 13
2.11 Startup Latency [3] 13
2.12 QUIC Connection Migration [2] 15
2.13 Top-level Structure of Qlog File [20] 16
2.14 QUIC Events used in Qlog [21] 17

3.1 Topology of the Experiment 19
3.2 QUIC Connection Setup 21
3.3 SSLKEYLOGFILE Imported to Wireshark 22
3.4 Wireshark Capture of Connection to Public QUIC Server 22
3.5 Wireshark Capture Showing Encrypted QUIC Packets 23
3.6 Wireshark Capture Showing Decrypted QUIC Packets 23
3.7 Diagram of Laboratory Setup 25

4.1 Baseline Wireshark Capture 28
4.2 Result of Baseline Capture 29
4.3 GEO Wireshark Capture 30
4.4 Result of GEO Capture 31
4.5 LTE Wireshark Capture 32
4.6 Result of LTE Capture 32
4.7 QUIC and HTTP/3 Visualization Toolsuite 33
4.8 Sample Qlog Trace 34
4.9 Qlog Stats 35
4.10 Qlog Stats 36
4.11 QVIS - Congestion Tool 36
4.12 QVIS - Multiplexing Tool 37

4.13 QVIS - Packetization Tool 37
4.14 QVIS - Sequence Tool 37
4.15 QVIS - Sequence Tool 38

GLOSSARY

CHLO	Client Hello
CLI	Command Line
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
FEC	Forward Error Correction
FreeBSD	Free Berkeley Software Distribution
FTP	File Transfer Protocol
GEO	Geostationary Equatorial Orbit
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
LaTeX	a document preparation system for scientific writing
LTE	Long-Term Evolution
RTT	Round Trip Time
SHLO	Server Hello
SSH	Secure Socket Shell
SSL	Secure Sockets Layer
TAU	Tampere University
TLS	Transport Layer Security
TUNI	Tampere Universities
UDP	User Datagram Protocol
URL	Uniform Resource Locator

1. INTRODUCTION

1.1 Overview

With the invention of the Internet, protocols have become the most essential part of the Internet – a building block. A computer protocol is used to execute technical standards based on certain rules [10]. While network protocols form the set of rules that govern the way data is transmitted between devices connected in a network. Examples of some protocols (network) are, Hypertext Transfer Protocol (HTTP), QUIC protocol, File Transfer Protocol (FTP), Secure Socket Shell (SSH). Communication, network management and security are the primary actions of network protocols [5].

In communication, the protocols enable the connected devices in the network communicate with each other, transfer files, or access the Internet, such as QUIC protocol, File Transfer Protocol (FTP), Internet Protocol (IP), Bluetooth, Instant Messaging. Network management protocols guarantee the whole networks performs ideally, by ensuring the individual devices like computers, routers, servers, function optimally. They can establish connections; combine multiple network connections between devices, into one link – link aggregation; troubleshoot networks. For security protocols, they protect the data or network it sends from unauthorized access. Some of its basic functions are encryption, authenticating entities, protecting data in transport [5].

According to [8], the choice of protocol is very vital in a communication system. In this paper, the web is the communication system where devices and networks are interconnected. The underlying protocol of the web is the HTTP, known as "the protocol of the web" [29]. HTTP is used solely for communication over the web, in the transfer of files, video, text, other multimedia files. Video content is the largest Internet traffic source, and this propelled the use of HTTP on the web [3].

The weaknesses of HTTP were barriers to a faster Internet, even if studies showed that HTTP would be the future of the Internet [3] – which it was for about 15 years, until the arrival of QUIC protocol. QUIC protocol was developed to improve the overall performance of the Internet – by improving HTTP traffic performance, due to emergence of latency-sensitive applications and demands in bandwidth, which was unsustainable for HTTP [4].

In this thesis, QUIC traffic was inspected, and the feasibility of the tools used in this inspection. These tools are used to show the behavior in different networks, and further compared.

1.2 Research Questions

This thesis has three distinct research questions:

1. What is the current state of QUIC protocol?
2. How does QUIC protocol perform in networks, and the feasibility of the tools used in inspecting QUIC?
3. What security does QUIC protocol offer?

The second research question is the focus of this thesis. The rest of the questions would be answered for more insights into QUIC protocol, and build a solid structure for the thesis.

1.3 Scope

The scope for research question 1 will explain how QUIC came about, the current state of QUIC, and possible shape of its future.

The scope for research question 2 which happens to be the focus of this thesis, will describe how QUIC performs under different network conditions, a baseline network condition, Geostationary Equatorial Orbit (GEO) satellite network condition and Long-Term Evolution (LTE) network condition. It will also explore the tools used in capturing QUIC traffic and evaluating its performance in the traffic.

The scope of research question 3 will highlight the different properties of QUIC and how they come together to form its security properties. Also, it briefly describes the susceptibility of QUIC to attacks or the attacks QUIC can protect the average user from.

1.4 Methodology

The purpose of this thesis was to provide detailed background information about QUIC, its properties, which form its advantages, and disadvantages. Also, to evaluate QUIC's performance under different network conditions, and the feasibility of the tools used in the inspection.

The detailed background information of QUIC was carried out by researching journals, papers, websites and blogs; using Google scholar, Andor – Tampere University academic search engine, and Google search engine.

The evaluation of QUIC's performance under different network conditions was carried out by setting up QUIC server, uploading documents to the server, then using wireshark to capture the traffic. The Secure Sockets Layer (SSL) session keys were also imported to wireshark to be able to decrypt parts of the QUIC traffic. The captured traffic on wireshark was illustrated using qlog and qvis as visualization tools.

1.5 Structure of the Thesis

This thesis is divided into five distinct parts;

- Introduction
- Background
- Design and Implementation
- Results
- Conclusion

Each of these parts of the thesis are intertwined together to form the whole picture.

The first chapter of this thesis is the introduction of the topic. The introduction sets the pace for the thesis, telling the reader what the thesis will be about. Describing the research questions, the scope, methodology of the thesis, and the structure of the rest of the thesis.

The background part is Chapter 2 of this thesis. This gives a brief history of HTTP, describes how and why QUIC came about, the improvements made to the HTTP stack. It further describes the current state of QUIC, the properties of QUIC. In describing the properties of QUIC, the advantages and disadvantages are discussed, which also highlights the security properties of QUIC, and its susceptibility to attacks or the attacks that can be prevented whilst using QUIC protocol. Then the feasibility of the tools used in measuring its performance.

The third and fourth chapters are about the design and implementation part of thesis. In details, it describes the setup of the environment to carry out the evaluation of the traffic, how the capture and measurement will be done – using wireshark [28], the feasibility of the tools used for evaluation, the visualization of captured traffic with qlog and qvis tools [24], and the networks in which they performed.

The conclusion part of the thesis which is the fifth chapter, summarises the thesis as a whole, it reminds the reader of the salient points, basically the takeaways. The chapter also points out the limitations faced while carrying out the design and implementation, and suggests recommendations to the limitations. This chapters helps to make sense of everything read from the beginning of the thesis to the end.

2. BACKGROUND

2.1 HTTP

HTTP is an application level protocol – simply a protocol that defines how messages are transported across application processes, typically running on different end systems[3]. It is the most widely used protocol by web-based applications, it is used in building up connection and communication between end-points, remote servers [27]. It was invented by Tim Berners-Lee in 1989.

There are different versions of HTTP, which are HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP/2, HTTP/3. HTTP/0.9 and HTTP/1.0 are now obsolete HTTP versions.

The slow start in HTTP/1.0 was improved in HTTP/1.1 by applying just one Transmission Control Protocol (TCP) socket when downloading. To download a web-page, only one persistent connection was used. The concept of *pipelining* was introduced in HTTP/1.1, when a client that sends a request to a server, sends out many requests without waiting for a response from the server [27].

HTTP/2 was released as an improvement to its predecessor – HTTP/1.1. HTTP/2 utilized network resources effectively [4], by multiplexing multiple HTTP requests on the same TCP connection [11]. It also improved header compression [4].

HTTP/3 is known as HTTP over QUIC, because it uses QUIC as the transport layer protocol [1]. The idea was to make the best use of QUIC, to help improve the Quality of Experience of users and web performance [8]. This also implies that HTTP/3 connections can only be accomplished over QUIC [4].

According to [1], the following are notable features of HTTP/3:

1. **QUIC protocol:** It improves web performance with its faster connection establishment, the built-in encryption system and its congestion control.
2. **Connection migration:** As part of QUIC protocol, there is provision for connection migration. Users of HTTP/3 protocol can easily switch between networks, without a break in connection – mobile network to WiFi.
3. **Better security:** There is a secure communication between clients and servers provided by the built-in encryption system.

4. **User Datagram Protocol (UDP)-based transport:** Because HTTP/3 relies on QUIC protocol and QUIC protocol was built on UDP, by extension HTTP/3 uses UDP. This UDP translates to faster connection, lower latency but no reliable delivery, QUIC has its own reliability system.

2.2 Why QUIC?

As previously stated in the introductory chapter, [8] makes us understand that, the choice of protocol is the most essential part of a communication system – the web. It was paramount to have a protocol to improve the performance of web traffic and make it more secure. Most of the web is powered by HTTP/2.0, with Transport Layer Security (TLS) on top of TCP. A downside to this structure has been the emergence of latency-sensitive applications and demands in bandwidth, which was unsustainable for it [4]. It is worth noting that QUIC was envisioned for web-content delivery and mobile applications [16]. [13] established QUIC as the foundation of HTTP/3.

2.3 What is QUIC?

QUIC protocol, developed as an advancement for HTTP(s) stack of HTTP/2, TLS, TCP; was proposed by google and standardized by the Internet Engineering Task Force (IETF) [8]. It was formally standardized by the IETF in May 2021, but its initial experimental version dates as far back as 2012 [15], it was standardized in RFC 9000 [4], and QUIC is not an acronym.

It is a transport protocol built on UDP [8]. By building QUIC on UDP instead of TCP, the round trip time of TCP handshake is eliminated. Also, an encryption system is infused in QUIC – which is similar to TLS, connection establishment and key generation jointly make up one round trip – which can sometimes be zero round trip [6]. In achieving this reduced round trip, connections use known server credentials for recurrent connections, which eliminates unnecessary handshake overhead in the network stack (reduces handshake latency), which is called cryptographic handshake [15]. For faster development, the *reliability* of TCP in the transport layer is implemented in the application layer [23].

There are two versions of QUIC; the gQUIC – the original protocol that was designed by Google, and IETF QUIC – now regarded as just *QUIC*, adopted from gQUIC with a lot of modifications [11]. gQUIC transports HTTP/2 frames, while IETF QUIC transports application protocols such as HTTP/3, DNS – making it general purpose, this is a pratical difference between the two versions of QUIC [4].

In this thesis, QUIC here would be referring to the IETF standardized QUIC, while any reference to gQUIC will be *gQUIC*.

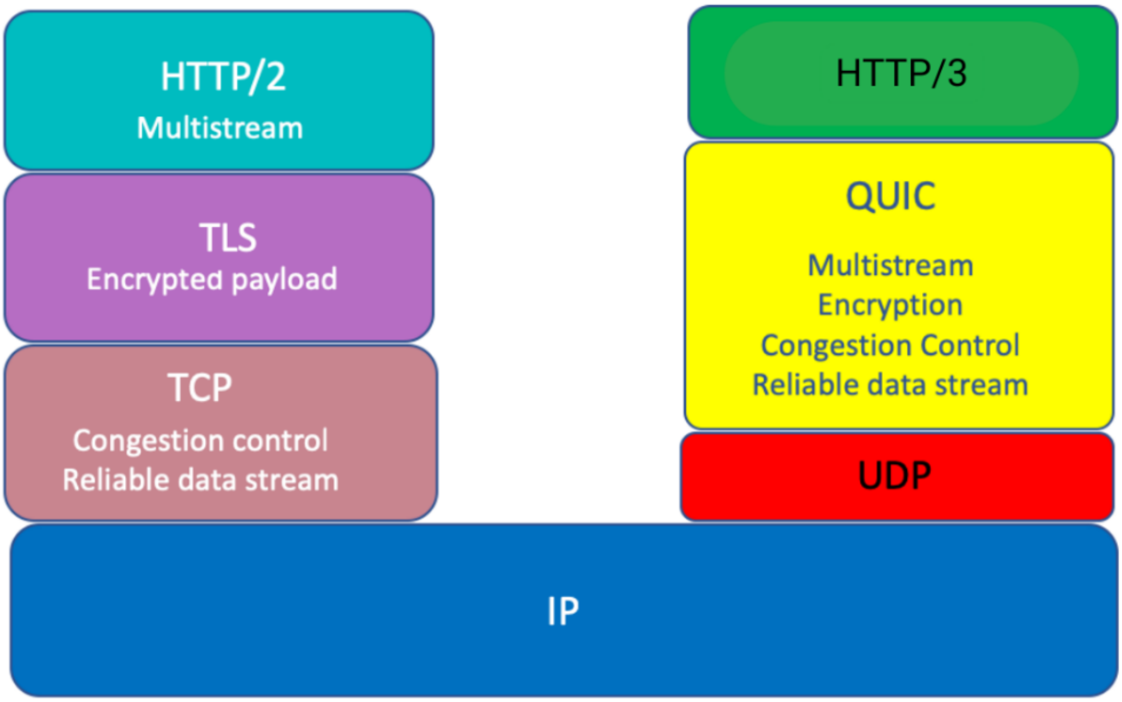

Figure 2.1. *Architecture of QUIC Protocol [12]*

QUIC packet: this is contained in UDP datagrams which are usually exchanged by QUIC endpoints. Long and short headers are the types of QUIC packet headers. Long header defines packet headers that have the most significant bit of first byte set, and short headers do not have the most significant bit set – the bit is cleared [26].

In 2.3, according to [26], the most significant bit of first byte is *header form* and is set to 1. While in2.2, the *header form* is set to 0, which signifies that it is short header, which also does not contain Destination Connection ID Length, Source Connection ID Length, Source Connection ID and Version fields.

```
Short Header Packet {
  Header Form (1) = 0,
  Version-Specific Bits (7),
  Destination Connection ID (..),
  Version-Specific Data (..),
}
```

Figure 2.2. *Short Header [2]*

The header structure of QUIC is as follows:

```
Long Header Packet {
  Header Form (1) = 1,
  Version-Specific Bits (7),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Version-Specific Data (..),
}
```

Figure 2.3. *Long Header [2]*

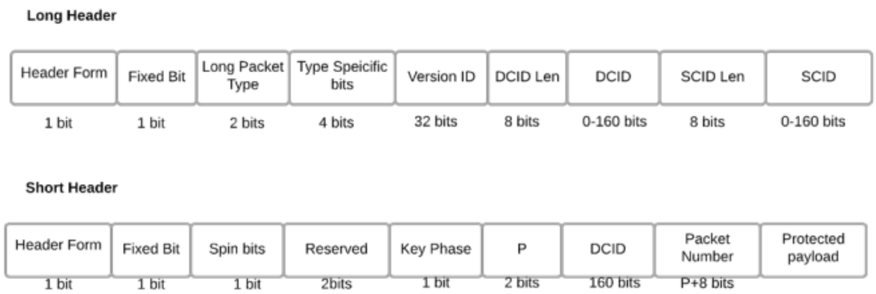

Figure 2.4. *QUIC Header [2]*

Public flag: the existence of connection ID field and length of the packet number field are concealed by the flag

Version: This has an identifier 4-byte in size, and is used in associating the QUIC protocol to its version, identified by the endpoint [26].

Connection Identifier: connection ID is used by QUIC connection to deliver packets to the right endpoint, as a result of address change at lower protocol levels – UDP, IP. It has to be unique for every connection, to prevent packets from being sent to the wrong endpoint. Both the source and destination have separate connection IDs [2].

This connection ID is used in mapping packets to its resulting QUIC connection [26]. Connection Identifier is not based on IP address, therefore, any change in network is conveniently handled between the networks, and eliminating the need to establish the connection again [3].

Version Negotiation: a version negotiation packet is triggered, and sent as a response in a case where a long header type of packet is sent to a QUIC endpoint, with a QUIC version that the QUIC endpoint does not understand. Version negotiation packet is in line with the format of long header type of packet, it does this by setting the high bit of the first byte. It is not triggered by a short header type of packet, and is identifiable by the version field [26].

```
Version Negotiation Packet {
  Header Form (1) = 1,
  Unused (7),
  Version (32) = 0,
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Supported Version (32) ...,
}
```

Figure 2.5. *Version Negotiation [25]*

Packet number: this identifies cryptography nonce, which protects packets. For all traffic – incoming and outgoing, and for each endpoint, the packet number differs. This packet number is also segregated into 3 spaces by QUIC: *initial space*, where all initial packets are contained; *handshake space*, it encompasses all handshake packets; *application data space*, this is where the 0-Round Trip Time1 (RTT) and 1-RTT packets are held [2].

Figure 2.6. *Header Structure of QUIC packet [26]*

2.4 Properties of QUIC

2.4.1 Security

According to [4], QUIC protocol is very enticing to threat actors due to the rising traction gained, and its acceptance among major technology organizations.

QUIC protocol is an encrypted-by-default transport protocol [11]. To develop the security and congestion control functionality in QUIC, it incorporates parts of TCP, TLS, and Datagram Transport Layer Security (DTLS) [16], this allows for reliable communication [8]. For QUIC, TLS 1.3 is the security component [4].

Naturally, application of security is through authenticated and encrypted packet header and pay-load, this prevents modification [6]. Cryptographic protection is added to gQUIC, this protects against Internet Protocol (IP) spoofing and packet re-ordering. IP spoofing as defined by Microsoft, is when threat actors maliciously attack devices with malware, to crash your server or possibly steal your data. Packet re-ordering and drops is solved by including a packet number in each packet [4]. [25] explains that, by encrypting everything in the transport layer, this eliminates the dependence on TCP + TLS mode of encryption. In turn, modifications by interceptors is prevented, which all improves the security in QUIC.

The built-in security of QUIC protocol improves the performance of QUIC protocol. The three-way handshake in TCP and the TLS 1.3 handshake are combined into QUIC's initial handshake. This causes the initial connection establishment to be faster, leading to just a single round trip between client and server, in the initial handshake [11]. The TLS v1.3 provides some security against handshake Denial of Service, because the security of handshake procedure is dependent on the TLS v1.3 handshake [4].

A downside to this 0-RTT connection establishment can be the replay attack from previous connection handshake. This replay attack can be advanced to a Denial of Service (DoS) attack, if the client and server are made to complete a handshake, and it leads to computational resources and memory space to be used up [7]. The threat of pervasive monitoring attacks is a possibility with some information such as connection identifier, because of its unencrypted state [7].

[9] discussed some common security issues with QUIC and how they are addressed, they are as follows:

- Handshake costs: for QUIC, each connection is uniquely encrypted. To achieve this, the server provides the configuration, and the configuration supplies the public keys used to encrypt the communication.
- Replay attacks: the client sets a unique nonce which QUIC uses to avoid a request getting replayed later – a unique nonce per request processed. Nonces are stored in a shared register by the server, which is checked when requests are received, and if a nonce exists, such requests are rejected.
- Wire Protocol: this helps overcome traffic analysis problem, by including padding (PAD) tag to message data, which keeps the bandwidth consumption constant at all times.
- IP address spoofing: to protect devices using QUIC protocol from IP spoofing, a source-address token is used. This source-address token is encrypted client IP and server timestamp, which is sent by the server to the client IP only. The token helps to prove ownership of their IP in subsequent requests that will be made to the server.

[11] describes with 2.7 the way HTTP request between client and server, is carried out over TCP and TLS. While 2.8, [11] describes how HTTP request is carried out over QUIC.

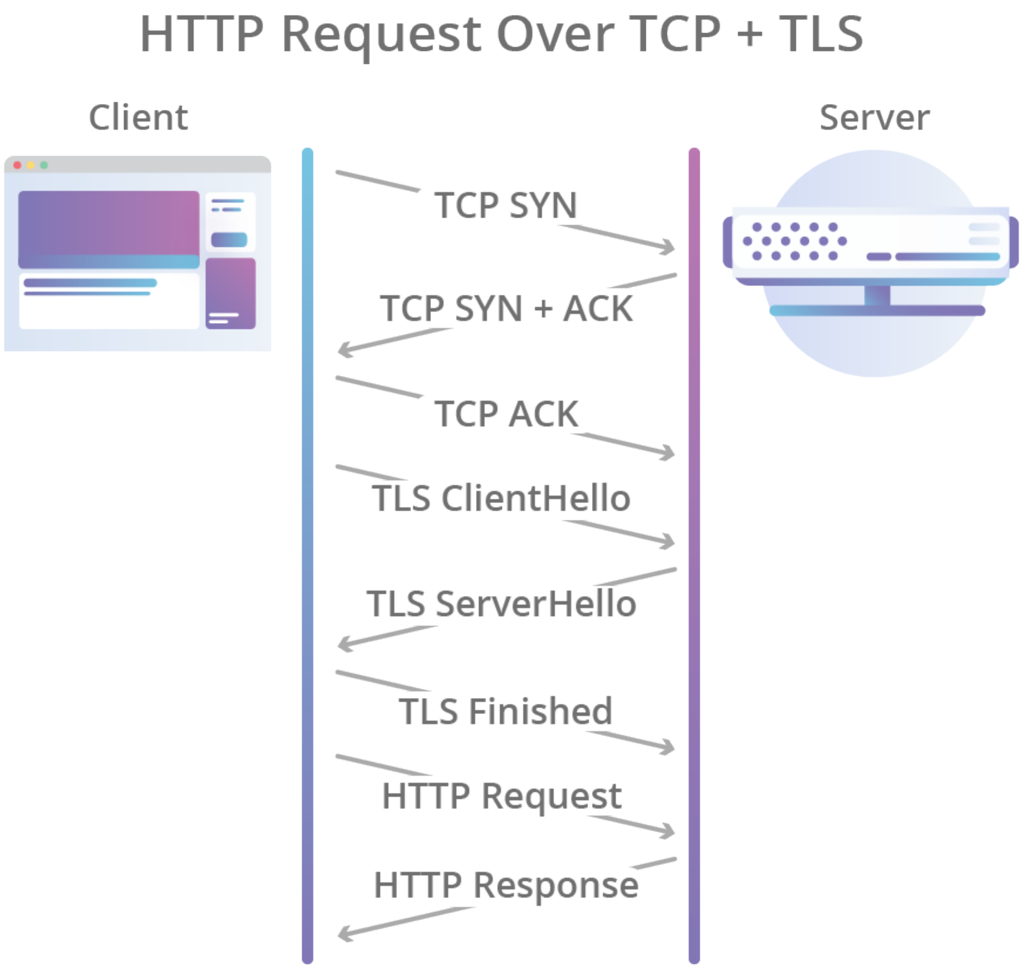

Figure 2.7. *HTTP Request over TCP + TLS [11]*

2.4.2 Stream Multiplexing

In HTTP/2, head-of-line (HoL) blocking was a problem at the TCP level with connections, when a stream is held up up because of a lost or delayed packet, where multiple simultaneous streams exist between endpoints communicating [8].

QUIC allows a client send many HTTP requests over same UDP socket, and in turn receive many responses on the same UDP socket, this is known as *multiplexing*. Simply allows endpoints that are communicating to have multiple concurrent streams exist between them, while eliminating head-of-line blocking. There is premium support for multiplexing by QUIC [11]. With this property, there is better organization of streams, use of the same UDP connection for more traffic, and allows for the compression of HTTP headers in the same connection [3]. In [22], header compression causes a significant

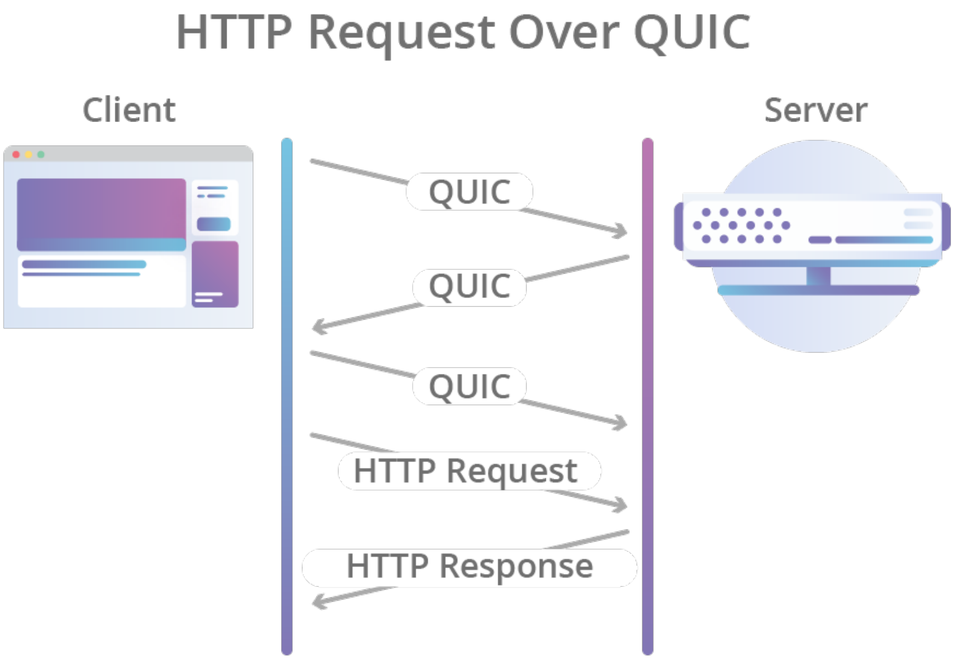

Figure 2.8. *HTTP Request over QUIC [11]*

reduction in needless header information, whenever there is a new page request.

2.4.3 Lower Latency

QUIC was designed to provide lower latency and security [13]. QUIC takes the best of UDP and TCP, to realize its latency reduction capability on the Internet [22] .

Low latency is achievable in QUIC because it is built on UDP [6]. TLS encryption handshakes and connection establishment are integrated together, and simplified in QUIC; this aids faster connection, re-connection [8]. This makes the connection time to be one round trip – for a fresh connection, and zero round trip time (RTT) if there was already a pre-established connection [3]. The inception of wireless devices has been a major driving factor for the need of faster access times of the Internet[22].

First time connection establishment: This connection is established after a successful version negotiation, and one RTT is used in the establishment [7]. The first time a connection is initialized, there is no information about the server with the client. But once the handshake has succeeded, the server information gets stored by the client. For the first time connection, the client sends a client hello (CHLO) message to the server. Then the server sends a reject (REJ) message, which holds information about the authentica-

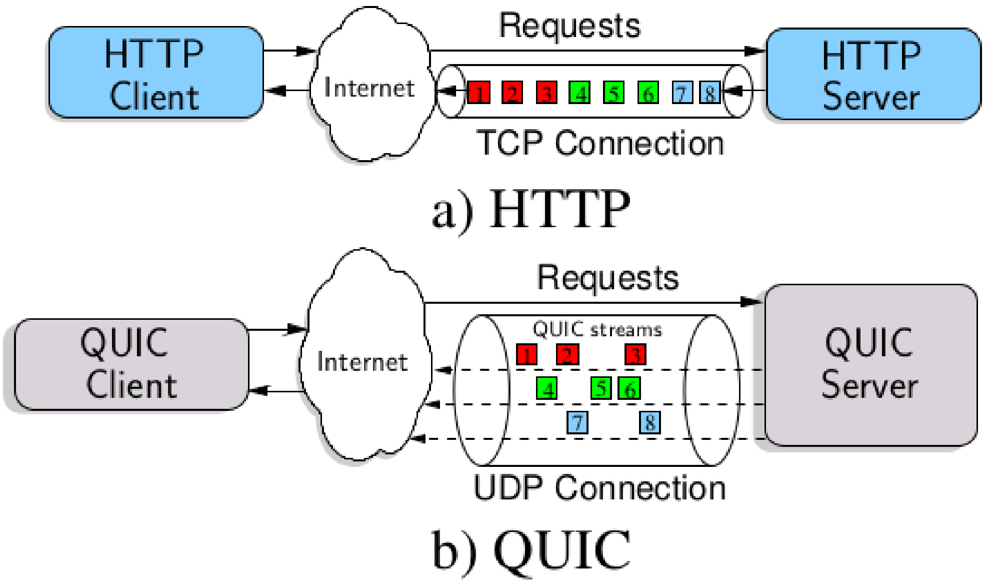

Figure 2.9. *Multiplexing [3]*

tion certificate, source-address token, and signature for the server certificate. This reject message is used in forming a complete client hello message, which is eventually sent to a Diffie-Hellman public key that is temporary, for the client [2]. It is important to note that, if the version of the client is not supported by the server, the client will be forced to go through an additional version negotiation process. Or the client receives the Server Hello message, certificate, session certificate from the server, which will now be used for subsequent connection to the server [7].

The zero round trip time is accomplished with the initial keys for the connection, from the initial handshake. The client can begin to send data to the server before it receives Server Hello (SHLO) message, immediately after it sends a complete CHLO message. When the client receives the SHLO message, it begins to use the final keys calculated from the details of the SHLO message to send data [2].

According to [7], in the course of establishing the first connection, there are parameters negotiated and are stored on the client, contained in a cryptographic cookie. The encryption key of the QUIC protocol is calculated with Diffie-Hellman value, this is also contained in the cryptographic cookie. All these information serves as the foundation for establishing 0-RTT connection.

In 2.10, [2] used the diagram to explain connection establishment in QUIC, where the cryptographic and transport handshake have been infused together, to set-up a transport connection.

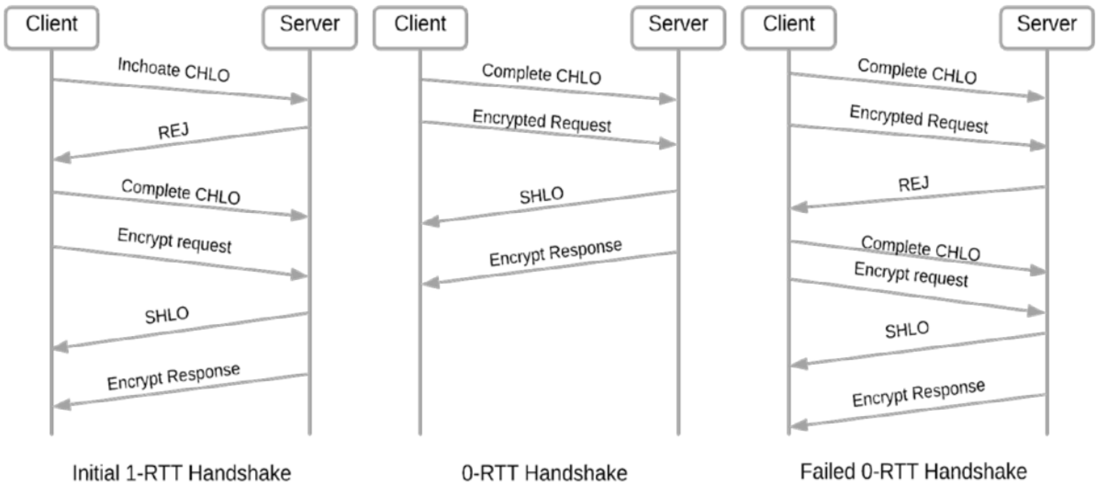

Figure 2.10. *QUIC Connection Establishment [2]*

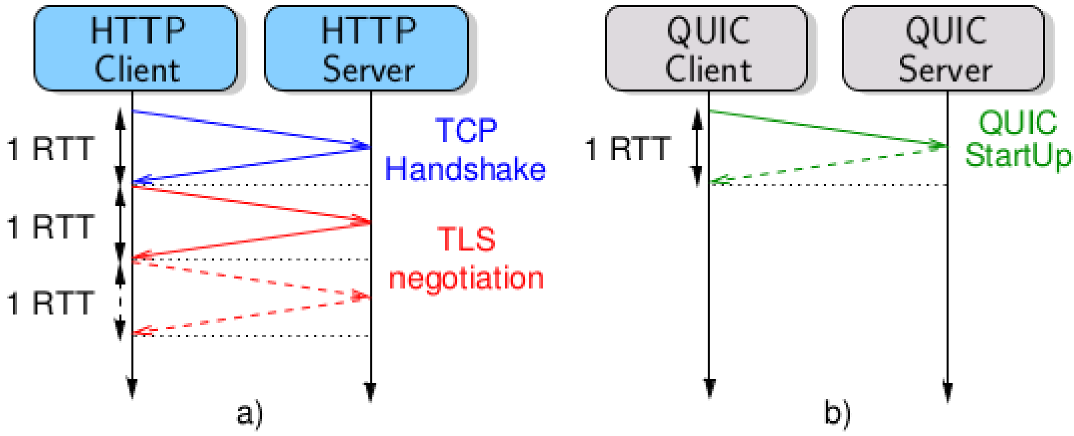

Figure 2.11. *Startup Latency [3]*

2.4.4 Congestion Control

With the support of multiplexing by QUIC, the same connection is shared by streams, extra handshakes are not required, and causes congestion to be shared. Packet loss of one stream does not affect other streams, because QUIC streams are delivered independently [11]. [15] tells us that QUIC protocol does not depend on any specific congestion algorithm.

2.4.5 Flow Control

Juniper Networks defines flow control as a mechanism for controlling traffic flows, in order to prevent frames getting dropped during congestion, it basically aids lossless transmission. The buffer size of the receiver is limited by *flow control*, when data is read slowly from QUIC's receive buffers, by an application.

We see in [15] that connection-level control and stream-level control are implemented in QUIC. In *connection-level control*, the total buffer consumed by a sender at the receiver across all streams is restricted, while the buffer consumed by the sender on any stream is restricted by *stream-level control*.

It is worthy to note that, the whole connection's receive buffer can be consumed by a slowly draining stream, which can hinder a sender from sending data on other streams, which is *head-of-line blocking*. By limiting the buffer size that a stream can consume, it reduces the risk of head-of-line blocking occurring [15].

2.4.6 Forward Error Correction

This is used for latency re-transmission reduction. Lost packets are not obviously re-transmitted, the receiver regains the lost packets by using redundancy in the sent data stream [6]. In managing packet loss, it swiftly recovers the lost packets, thereby making Forward Error Correction useful in head-of-line reduction over a QUIC stream [3].

Improving loss tolerance and quicker recovery with the use of any unnecessary information sent with Forward Error Correction (FEC) is done by coding – which detects and corrects limited number of errors in data transmitted. Coding is used when there is congestion and the packet loss leads to extra delays from slow recovery, replicated acknowledgements or re-transmission timeouts, especially after the sending rate has been correctly reduced. It is essential in situations where there is high packet loss – wireless networks, because it will boost performance, but the energy consumption will be more [7].

2.4.7 Connection Migration

A connection ID is used by QUIC to identify a connection, this is to help any connection between a client and server remain alive, if there happens to be subsequent change of port or client IP. A connection migration begins when there is a change in the client's network, and the client makes use of the connection ID from previous connection to make the request to the server whilst utilizing a probing packet [2].

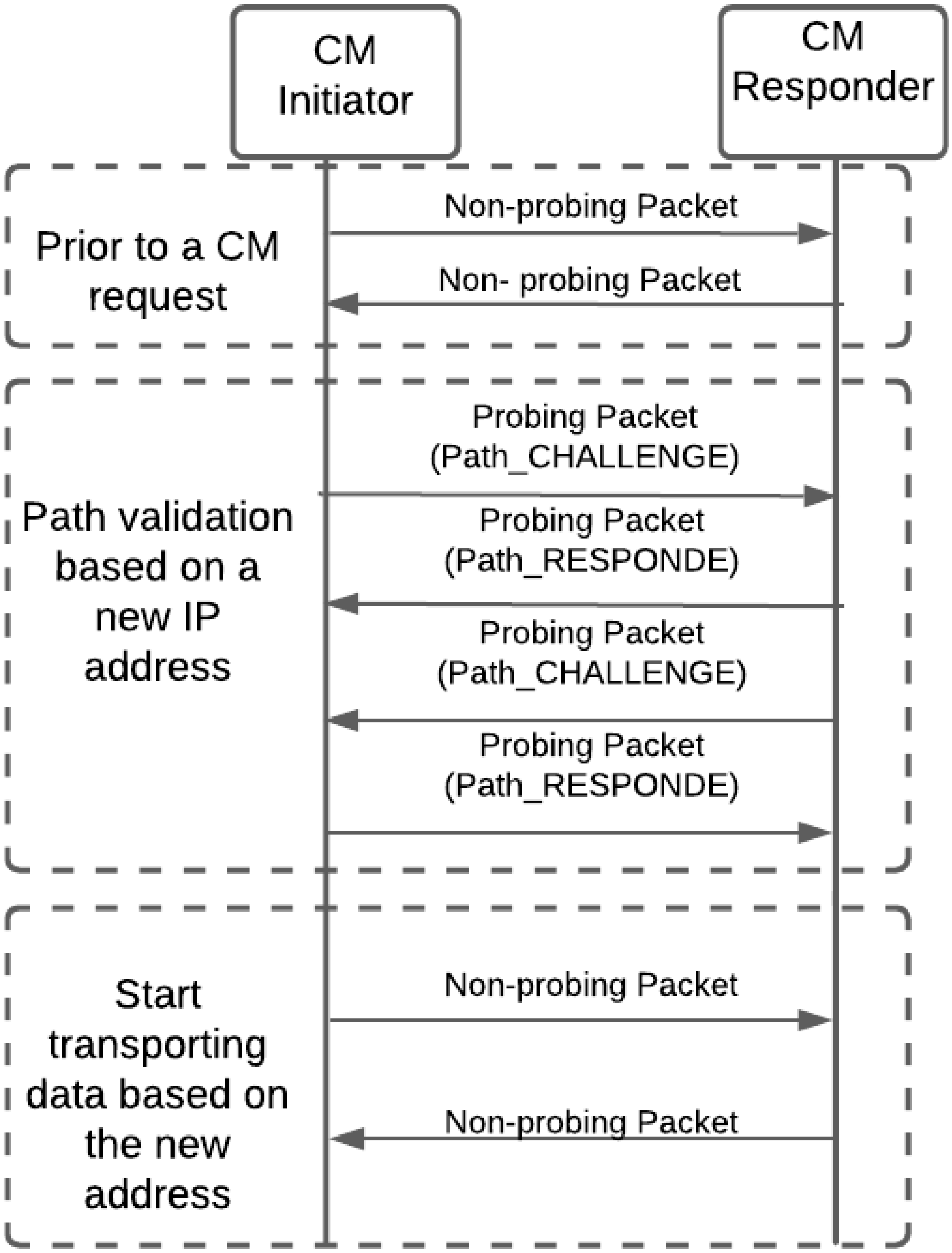

Figure 2.12. *QUIC Connection Migration [2]*

2.5 QLOG and QVIS

In order to secure QUIC packets, the packets are encrypted completely, except the fields used in routing, forwarding and decrypting the packet. This security feature has a downside, which is difficulty in analyzing and debugging the protocol – which are essential for fishing out bugs and understanding the behaviour of the protocol. Since the most important fields for analysis are encrypted, there is inadequate information contained in the metadata of packets. Furthermore, it is impossible to decrypt or encrypt packets in transit, because packets encrypted or decrypted in transit pose the risk of revealing the information contained in the payload. Also, session keys will be required accordingly. Where packets are sent from and received, are the best places to capture packets, these are the encryption and decryption points, *endpoints* [13].

2.5.1 QLOG

Qlog is defined by the Internet Engineering Task Force as an extensible high-level schema, that makes available a logging format for endpoints, that is whole, organized and shareable [20].

This logging format was created to help with the problems of encrypting and decrypting packets in transit, also, to extract sufficient information from packets that are not readily available in the packets' metadata. Qlog is built on JavaScript Object Notation (JSON), which makes it easily deploy-able, irrespective of language-specific characteristics. In order to be extensible, it has timestamp, category, event type, type-specific data aspects for each event logged [13]. The use of JSON allows for custom events to be defined, it is also easy to add, modify or extend new categories, metadata [18]. By tagging log details with high-level metadata, high-level filtering and tracing event chains can be done easily [19].

Some of the principles of qlog framework by [20] are:

- Event data and metadata are stored together
- It is stream-able, its logging style is event-based
- It is extensible
- It can be grouped and transformed easily

Figure 2.13 summarizes how the top-level of a qlog file looks like. *qlog version* describes header fields and component traces. *qlog format* describes the serialization method used, because qlog can be serialized in many ways, such as: JSON, CBOR, CSV. If the field is not set, "JSON" becomes the default option. It also helps qlog files parse better [20].

```
QlogFile = {
    qlog_version: text
    ? qlog_format: text .default "JSON"
    ? title: text
    ? description: text
    ? traces: [+ Trace /
                TraceError]
}
```

Figure 2.13. *Top-level Structure of Qlog File [20]*

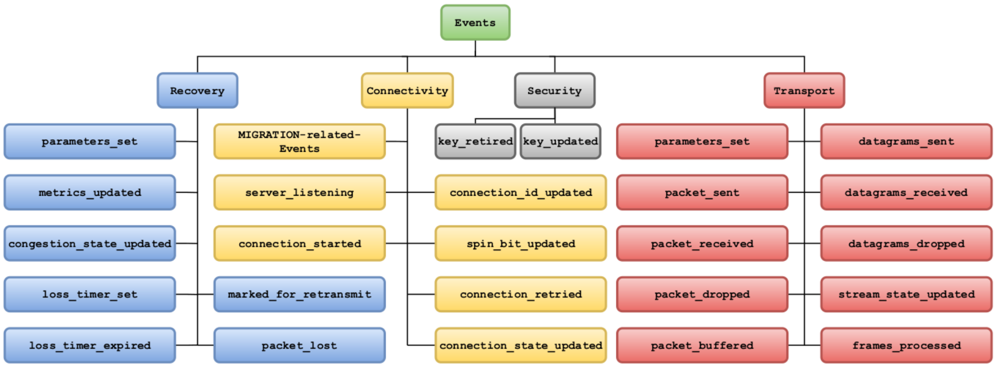

Figure 2.14. *QUIC Events used in Qlog [21]*

The structured and standardized format of qlog makes it important in the area of re-using existing tools, which qvis falls under [18].

2.5.2 QVIS

Textual formats – Qlog, could become impracticable when there are many long traces to be compared, and error-prone. By using qvis – an interactive visual tool, redundant details can be hidden and debugging is simplified [19]. Qvis visualizes qlog files and data, to make it clear and very illustrative, and can show information, such as round trip time. Also, it is designed to be used in browsers, and is largely executed TypeScript and Vue. [13] describes QUIC visualization methods, and they would be discussed in the following section:

2.5.3 Sequence Tool

This tool is used mainly to illustrate the flow of data from one endpoint to another, and is plotted on a vertical timeline. An interesting part of sequence tool is when it is used with logs from both client and server side, it is very accurate in displaying details of the log, such as: RTT, loss and reordering [19].

2.5.4 Packetization Tool

How QUIC packets are made up of QUIC and HTTP/3 frames is visualized by this packetization tool. In [17], packetization tool emphasizes the efficiency of wire-format.

2.5.5 Congestion Tool

It is used to illustrate in bytes, the amount of data sent over time, and the RTT taken by packets. Congestion window size and lost data can be shown with this tool. When there is a change in the slope of graph, it can be used to signify a change in rate of data being sent.

2.5.6 Multiplexing Tool

This visualizes how the sent data on the current QUIC streams were split on them. This tool in [17] helps to spot anomalies in patterns, to illustrate how Head-of-Line blocking affects QUIC, and it shows how the data re-transmissions in QUIC are organized.

2.6 Related Works

In [16], A security model was proposed to investigate QUIC's performance, to identify QUIC's strengths and weaknesses. In the course of the investigation, it was discovered that the very methods put in place for latency reduction, introduced security weaknesses for QUIC.

The model used in this work is aimed to be extended for use in other *performance driven protocols*. Also, the methodologies for dealing with the weaknesses were not explored, and will be relevant to be explored.

QUIC over UDP is examined in [3], by evaluating the web-page load time of different protocols – HTTP/1.1, SPDY and QUIC. The result of comparisons showed that QUIC had a better web page load time. Forward Error Correction, another property of QUIC, will make the performance of QUIC worse, when enabled.

This paper [18], assesses the application, utilization and dissemination of qlog and qvis – real world applications. Survey method was used among QUIC experts for the evaluation.

The use of endpoint logging for to replace packet capture in large deployments is not fully known, and open for future works.

In [21], QUIC and HTTP/3 were evaluated, based on their designs, benchmarks, for static and mobile cellular network. This assessment was done on real mobile networks, the data was recorded and collected from a platform.

The evaluation proved that QUIC was very efficient for connection migration i.e in users that move. Also, the congestion algorithm used was vital in affecting the performance of QUIC. With the tools used being applicable to future of QUIC's ever evolving nature.

3. DESIGN AND IMPLEMENTATION

3.1 Design and Implementation

In this chapter, the experimental setup to analyze the performance of QUIC protocol under different network conditions are described. We define the tools used to capture and analyze the traffic at different points, then the setups for the experiment were described.

3.2 Experimental Setup

3.2.1 Experiment Topology

For the analysis of QUIC protocol under different network conditions, 3.1 shows the simple topology of the setups. The setup is basically a client machine communicating with a server, over a network.

Figure 3.1. *Topology of the Experiment*

3.2.2 Software Tools

This section describes the tools that were used in setting up the experiment, actualizing the experiment topology, and capturing the QUIC traffic for inspection.

- **Qlog:** it was used to make available a logging format for the endpoints, that is whole, organized and shareable
- **Qvis:** it was used to visualize the qlog files gotten from the captured QUIC packets, to make it clear and very illustrative. It was used in the web browsers.
- **Operating System:** Linux was the primary operating system of this experiment, Ubuntu 22.04 was the distro and version. It was used along with the oracle Virtual-

Box virtual machine.

- **Wireshark:** this is a network packet analyzer and packet capture software [14], it was used to capture the live traffic of QUIC and HTTP/3 packets, to display data and meta-data of the captured traffic. The wireshark needed to be up to date, to be able to capture the packets. The wireshark version had to be installed using the source code from wireshark's own github repository. To launch wireshark the command *sudo ./wireshark* was run in the folder "run" located in "build" folder.
- **SSLKEYLOGFILE:** this is required in decrypting the QUIC packets. This environment variable instructs the browser to store the encryption keys into a specifically created file. This sslkeylogfile is used in combination with wireshark, which is used to show the decrytped information of the captured QUIC packets.
- **FreeBSD PC**: Free Berkeley Software Distribution was used to set the traffic conditions for the QUIC capture. It was used to set firewall rules, the latency, up-link and down-link speed, in the laboratory setup of the experiment.
- **Curl:** a command line tool used over network protocols for data transfer. It is used to upload files from the client to the server, and to download files from server to the client, over QUIC network – basically data exchange. The command line which follows is the syntax for single file upload: *curl -F fileToUpload='@'/path/to/file/ https://server-address/upload.php*
- **Web Browser:** Google Chrome was the web browser used in this experiment, it was the client in this experiment.
- **Virtual Machine:** Oracle VirtualBox was the virtual machine used in this experiment.

3.2.3 QUIC Client and Server

The QUIC client is a chosen web browser, which will establish the connection with the server, it will send the CHLO message in the handshake, For this experiment, the chosen web browser was Google Chrome, which already supports QUIC. QUIC is not enabled by default, to enable QUIC, the web browser had to be accessed from the command line interface (CLI) with these commands: *google-chrome –enable-quic –quic-version=h3-29*

For confirmation that the browser has enabled the QUIC and HTTP/3 connections, visit https://cloudflare-quic.com/. This would tell if the connections have been enabled by the browser.

The QUIC server sends the REJ/SHLO message during the handshake, and gives the client a certificate to verify. There are many public QUIC servers that use HTTP/3, available on the Internet such as: cloudflare-quic.com, google.com, www.litespeedtech.com, facebook.com.

3.2.4 Setting up the Connection

As the background of this paper described for setting up a QUIC connection, a handshake request is sent by the client to the server, after a successful version negotiation. It is worthy to remember that for a first time connection establishment, it takes one round trip, and for subsequent trips, it uses zero round trip. After a successful handshake, the client begins to send data to the server, which will be captured, this forms the basis for this experiment.

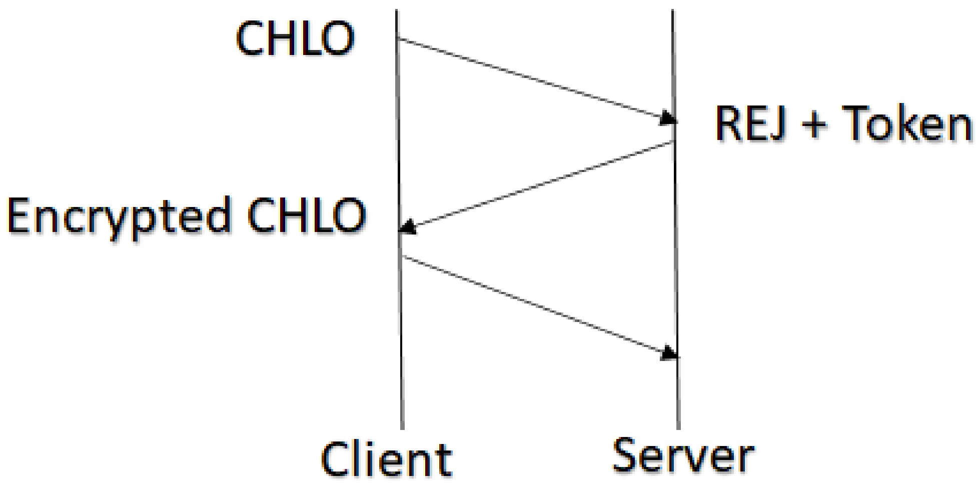

Figure 3.2. *QUIC Connection Setup*

3.2.5 The Setups

For the experiments to be actualized, there were necessary steps to be taken, set tasks, to actualize the implementations. There were two different setups used for this experiment. They will all be described in the following sections.

3.2.6 Setup A

For this experiment, the setup was done with the virtual machine connecting to public QUIC servers, and using wireshark to capture the traffic. For this setup A, the following steps were taken:

1. Installed the virtual machine with Ubuntu 22.04 on a Windows laptop
2. Next, the latest version of wireshark was installed from its own github repository.
3. Then SSLKEYLOGFILE was imported to the wireshark, to enable decryption of the encrypted packets. The information of the decrypted packets can be seen on wireshark when traffic has been captured. Fig 3.3 shows how the SSLKEYLOGFILE previously created was imported to wireshark, into the "pre-shared mastersecret."

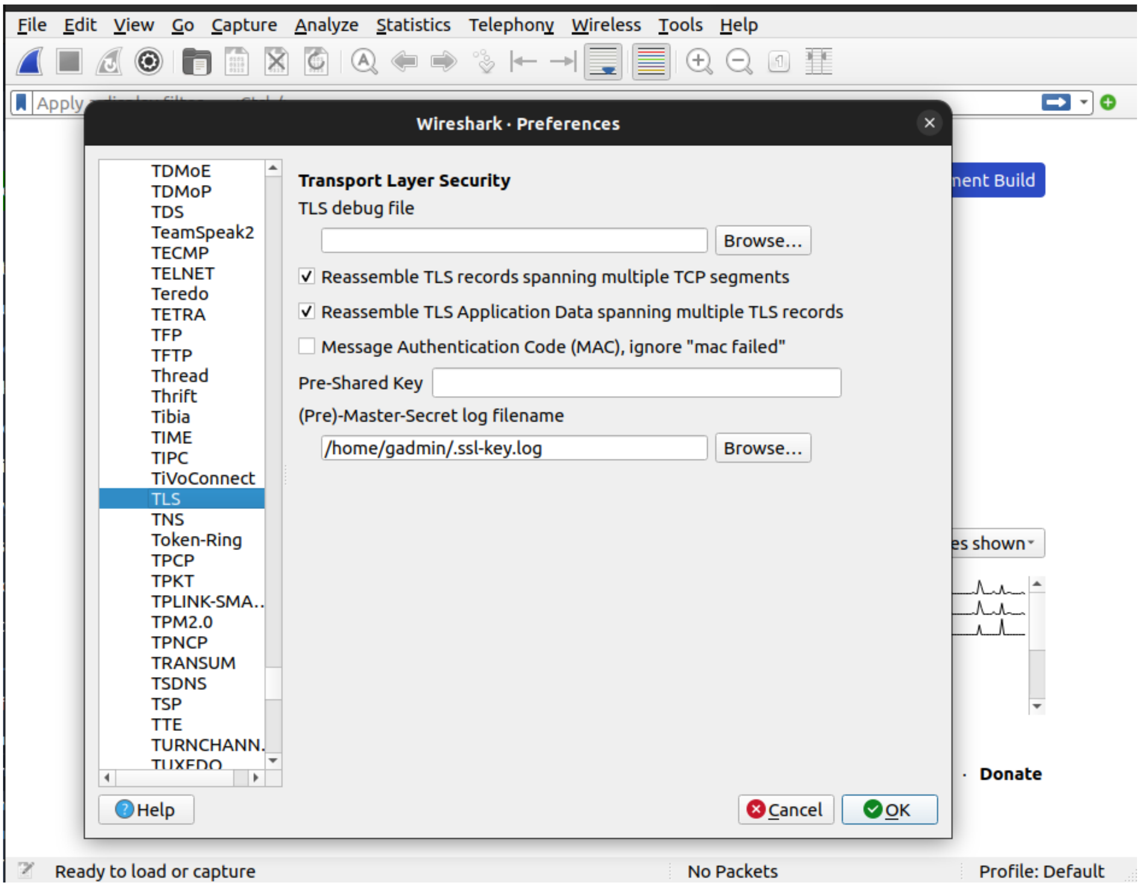

Figure 3.3. *SSLKEYLOGFILE Imported to Wireshark*

4.

In fig 3.4, the public QUIC server was *facebook.com* whose IP address is *157.240.205.35*.

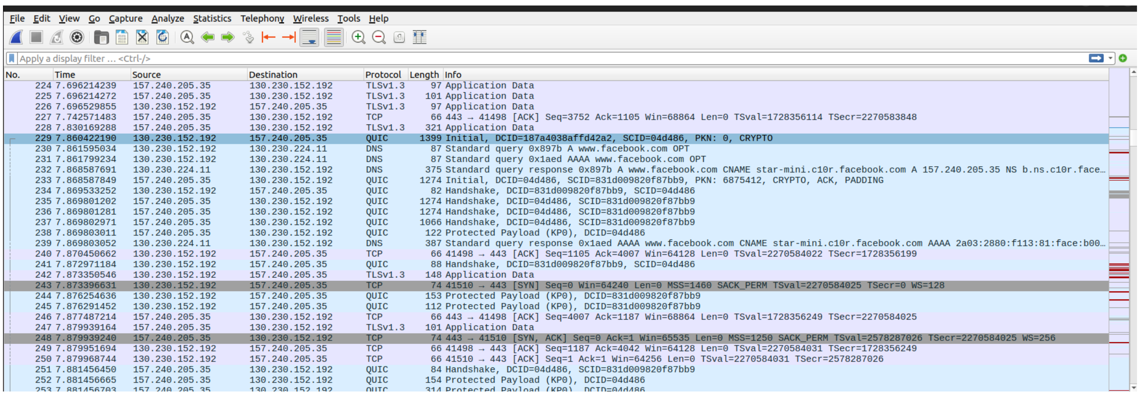

Figure 3.4. *Wireshark Capture of Connection to Public QUIC Server*

3.2.7 Setup B

This setup was a laboratory based setup as shown in fig 3.7, the QUIC server was connected to the cyberlab, through the RDNET. The cyberlab consisted of laptop, patch rack which housed the patch panel, switch and a FreeBSD PC (linksim). The FreeBSD PC and

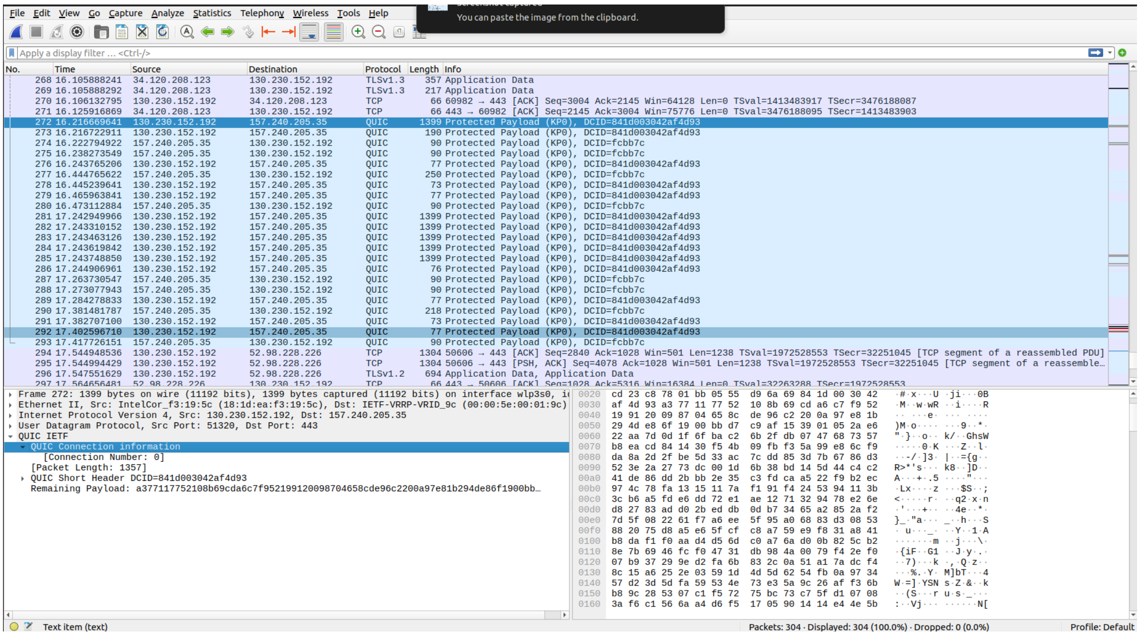

Figure 3.5. *Wireshark Capture Showing Encrypted QUIC Packets*

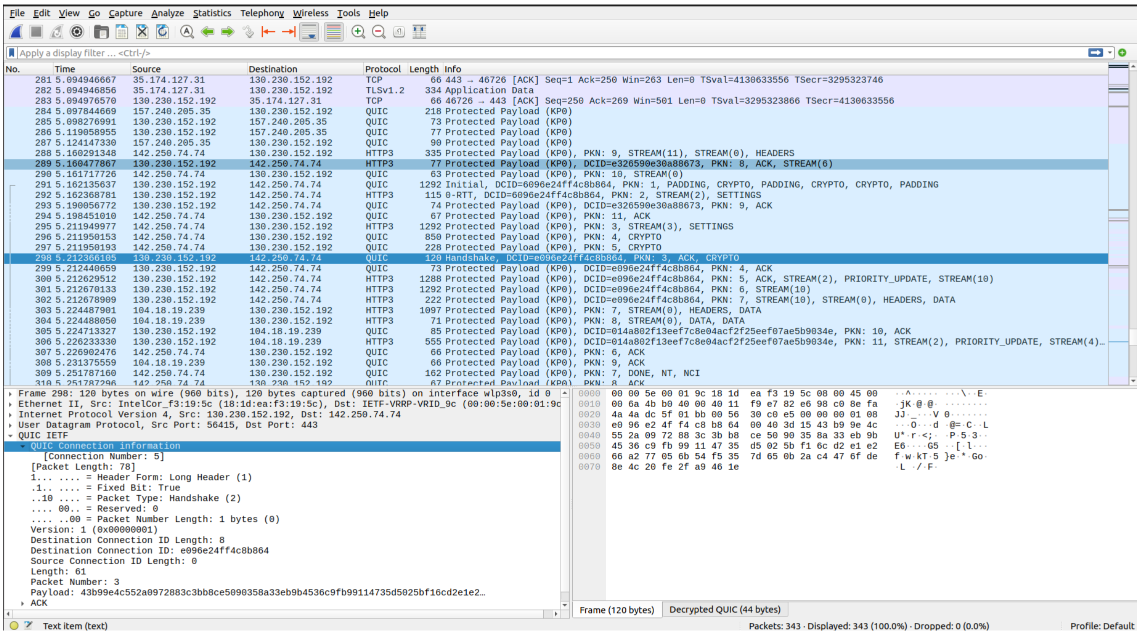

Figure 3.6. *Wireshark Capture Showing Decrypted QUIC Packets*

laptop were connected to the switch, then the FreeBSD PC was connected to the patch panel, the patch panel was now connected to the QUIC server linked by the RDNET. The QUIC server was openlitespeed server, hosted on another laptop. The client and server were on the same network when they communicated. The openlitespeed server used in this setup was a test server, because the experiment did not require anything sophisticated.

In this setup, the following steps were taken to be able to get the client and server to establish communication:

1. A laptop running Ubuntu 22.04 Linux operating system was the client's machine. Google Chrome was the browser which was used to connect with the server by sending the CHLO message.
2. For the server, the openlitespeed server was downloaded and installed using the instructions from their website. While the installation was going on, the admin credentials was given, which was important and used in subsequent steps.
3. When the installation was done, the server was started. The server's web admin console became accessible through the localhost:7080. To login, the admin credentials described in step 2 was used. This gave access to the configurations for the server and monitoring access.
4. The site is a test site by default, which was accessed using the localhost's address *127.0.0.1*.
5. In this step, configuring HTTPS connection was done. Because HTTP/3 and QUIC are encrypted by default, an authenticated certificate is required for it to work. The certificate used in this experiment for the local host was generated by *mkcert* – a tool for making trusted certificates locally. The instructions for installation were found on mkcert's github repository.
6. The certificate for the test site was created for the IP address of the machine hosting the server.

   mkcert localhost x.x.x.x was the command to create the certificate. It created *.pem and key.pem* files.
7. The two files created in step 6 were moved to the openlitespeed server's root folder */lsws/conf/cert*. Then a listener for port 443 (SSL) was created on the openlitespeed admin console, using the two files from step 6 – which were located in */lsws/conf/cert*.
8. The test site was now accessible through HTTPS connection.

After, successful completion of the steps, the client was able to establish communication with the server using curl. Curl was the tool used to upload files to the server. To evaluate the performance of QUIC under different network conditions, the evaluation is done from the client's point of view. The traffic characteristics are set with the FreeBSD tool, and the capture of the traffic data commenced.

3.3 Traffic Conditions

Following the lab setup and connection establishment between client and server, the FreeBSD PC was used to control the network parameters, ipfw was used to give the link simulator the traffic characteristics, to achieve the heterogeneous traffic conditions, which

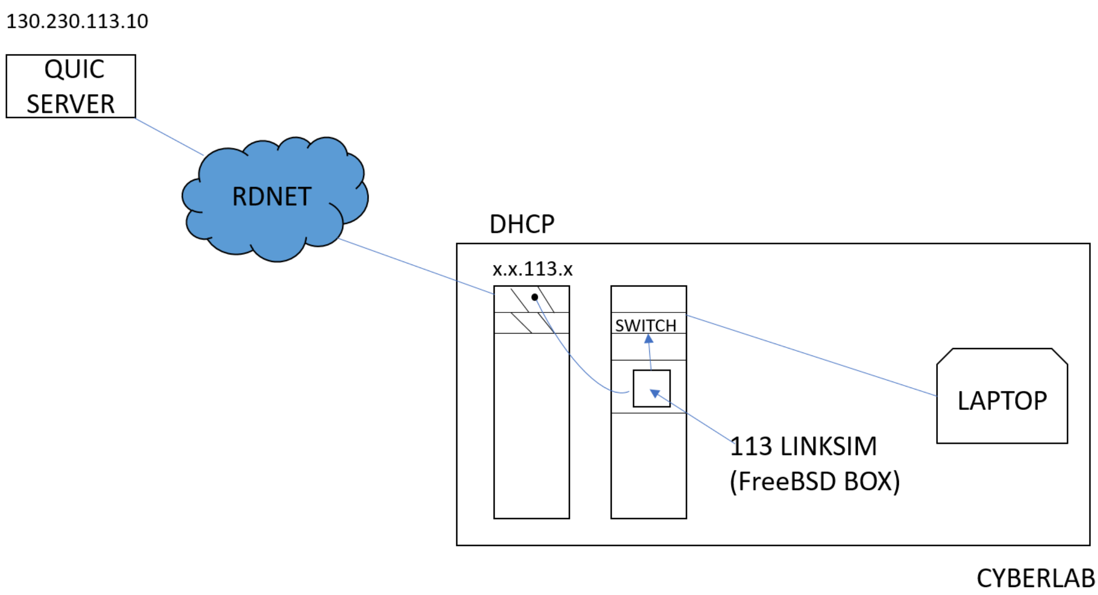

Figure 3.7. *Diagram of Laboratory Setup*

were:

- Baseline: in this traffic condition, there was no alteration with the FreeBSD PC, the client machine was connected directly to the QUIC server, which did not have any significant impact on the network.

  These were the traffic characteristics: *1000 Mbps full duplex, 0ms latency*

- GEO Satellite: this was achieved by using *ipfw* to set the traffic characteristics: *21 Mbps down, 3 Mbps up, 725ms latency*, to emulate a satellite network, as seen below:

```
#!/bin/sh

ipfw -q -flush
ipfw -q pipe flush

ipfw 0005 allow all from any to any via lo0
ipfw 0006 allow all from any to any via em0

#Upload

ipfw add 0010 pipe 1 all from any to any out recv em1
ipfw pipe 1 config bw 3Mbps delay 725ms

#Download
```

```
ipfw add 0020 pipe 2 all from any to any out recv em2
ipfw pipe 2 config bw 21Mbps delay 725ms

ipfw add 7000 allow all from any to any
```

- LTE: Long Term Evolution traffic condition was achieved using *ipfw* to set the following characteristics *236 Mbps down, 70 Mbps up, 10ms latency*, as seen below:

```
#!/bin/sh

ipfw -q -flush
ipfw -q pipe flush

ipfw 0005 allow all from any to any via lo0
ipfw 0006 allow all from any to any via em0

#Upload

ipfw add 0010 pipe 1 all from any to any out recv em1
ipfw pipe 1 config bw 70Mbps delay 10ms

#Download

ipfw add 0020 pipe 2 all from any to any out recv em2
ipfw pipe 2 config bw 236Mbps delay 10ms

ipfw add 7000 allow all from any to any
```

4. RESULTS AND ANALYSIS

This chapter discusses the results gotten from the design and implementation of this paper – chapter 3. This chapter will provide the reader a good understanding of QUIC's performance in networks, the tools used for inspecting the traffic and visualizing the captured traffic.

4.1 Capture of the Different Traffic Conditions

The wireshark capture and results of the network conditions set in 3.3 will be shown in the next sections. The results of the different traffic conditions shows the different time taken during data transmission, which are:

- time-namelookup: this column shows the time it took from the start of the connection for the name resolve to happen. The time was in seconds.
- time-connect: this column shows the time from the start of the connection taken for the TCP connect to the remote host was done. The time was in seconds.
- time-appconnect: this column shows how long the SSL/SSH handshake to the remote host took from the start, it was measured in seconds.
- time-pretransfer: this was the time taken from the start till the actual packet to be transferred was just about to begin. This was measured in seconds.
- time-starttransfer: this was the time it took from start of the connection till the first byte was going to be transferred. This was measured in seconds.
- time-total: this is the time taken for the whole connection lasted, including the complete data transfer. This was measured in milliseconds.

4.1.1 Baseline

Fig 4.1 shows the wireshark capture of the baseline traffic condition, while fig 4.2 shows the results of the capture with respect to time.

The performance of QUIC under this traffic condition has no latency.

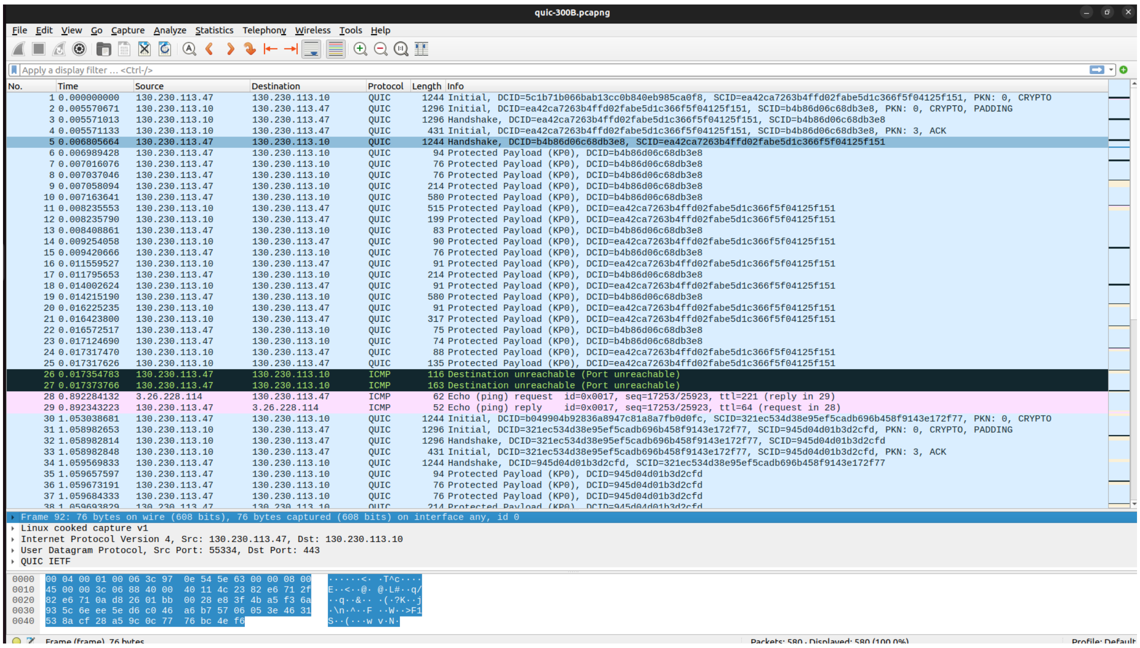

Figure 4.1. *Baseline Wireshark Capture*

4.1.2 GEO Satellite

Fig 4.3 shows the wireshark capture of the GEO satellite traffic condition, while fig 4.4 shows the results of the capture with respect to time.

The performance of QUIC under this traffic condition has little latency of 725ms, which is almost not noticeable during data transfer.

4.1.3 LTE

Fig 4.5 shows the wireshark capture of the LTE traffic condition, while fig 4.6 shows the results of the capture with respect to time.

The performance of QUIC under this traffic condition has a minute latency of 10ms, which is almost not noticeable during data transfer.

4.2 Visualization of Captured Traffic

In visualizing captured traffic, the pcap files obtained from the decrypted traffic captured by wireshark is converted to qlog events. For this paper, demo samples were gotten from qlog and qvis' github, and used for visualizations.

In fig 4.7, it shows the homepage of the visualization toolsuite for QUIC and HTTP/3 – quic and qlog. It is important to know that the only file formats supported by this toolsuite are: *.qlog, .sqlog, and .netlog*. On this page, there are different options for uploading the files of captured packets, the options are:

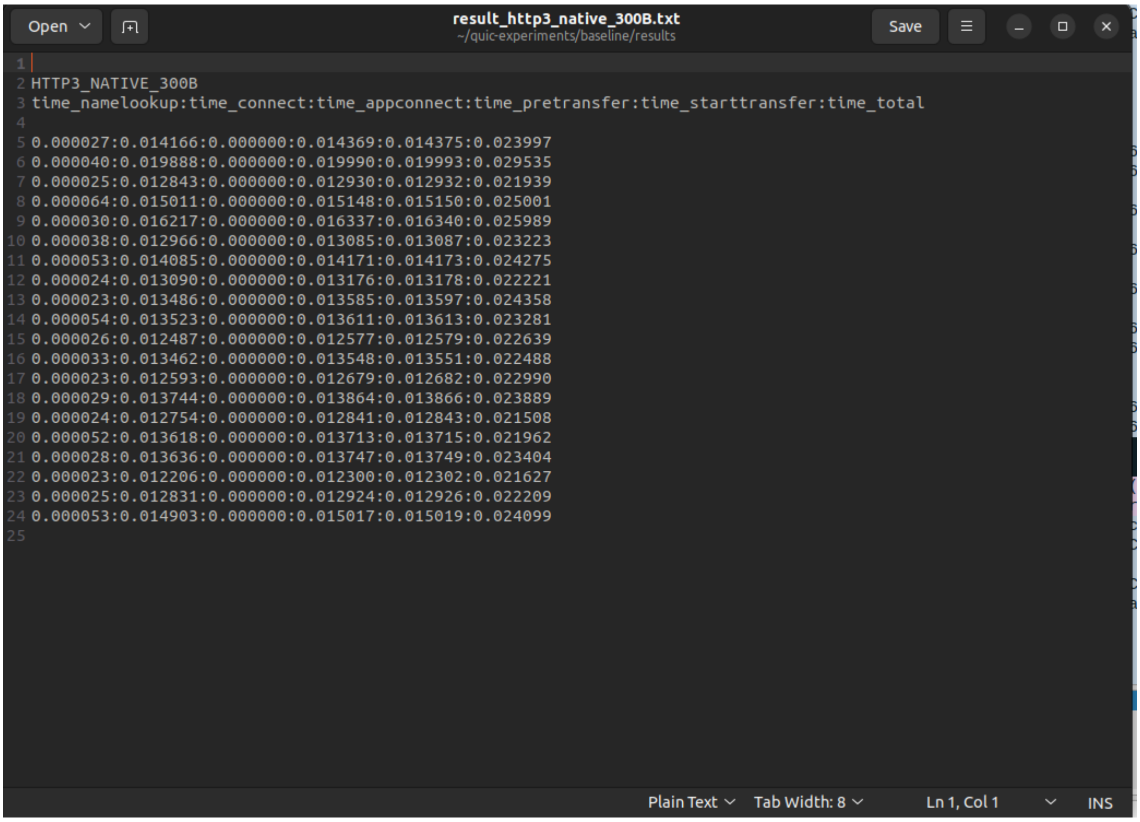

Figure 4.2. *Result of Baseline Capture*

1. Loading the file using URL: In this method, the link to the file is used for the upload, for visualization to be done.
2. Uploading the file directly: The file gotten from the packet capture can be uploaded directly in this method.
3. Loading demo files premade: Here, there are sample files from the qvis github repository that can be loaded and visualized.
4. Loading a massive demo file: This supports large single qlog files. It is used to show qvis visualizations on larger traces.

Also, on this homepage, the different qvis visualization methods are located here. After the file has been loaded, the different visualization methods can be viewed by just clicking on the particular one of interest. There is also the option of viewing *qlog stats*.

4.2.1 Diagrammatic Representation of the Visualization Tools

A sample of qlog trace file can be seen in 4.8, serialized in JSON, and it is in *.qlog* format. This is the file that was loaded into the toolsuite, and the *upload file directly* option was used to load the qlog trace file. In this trace file, the different data logged for the

In fig 4.9 and fig 4.10, the details about the traffic captured is given. This details are quite

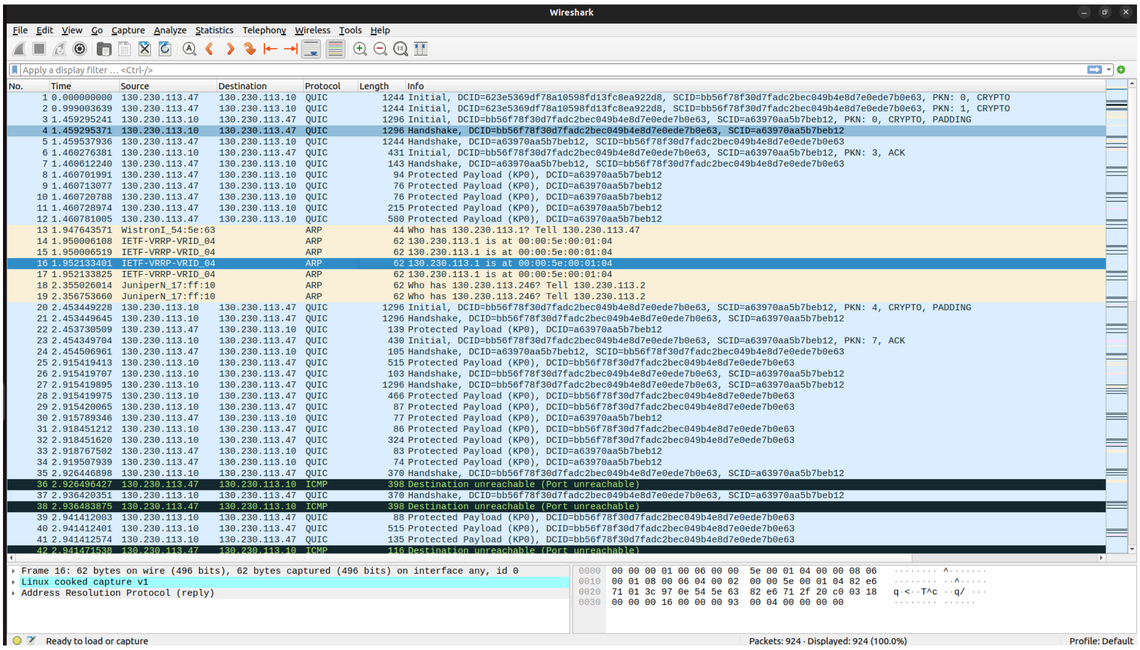

Figure 4.3. *GEO Wireshark Capture*

simple and readable here, but in a case where there are many traces captured, it becomes difficult to read. Details such as vantage point, events, frames, frame count, encryption levels, encryption level count, connection-level flow control evolution, stream-level flow control, can be found in the qlog stats.

Fig 4.11 shows a congestion tool, which is used to illustrate the data sent over the network – including re-transmitted data, which is measured in bytes. It also shows the round trip time taken by the sent data, which is measured in milliseconds. The color codes used in this figure are used to also indicate the data lost during transmission, data acknowledged, connection control flow control limit, and the sum of stream flow control limits.

Note that the data sent in bytes and round trip time in milliseconds are both plotted against time which is in milliseconds too.

The qvis multiplexing tool in fig 4.12 illustrates the count of stream frames received, which includes re-transmitted frames. It defines how the data sent over the network was split on the QUIC streams, congestion control property of QUIC. Each stream in the network are colored, but in this figure, there are less than five streams, so only the yellow color is distinct. In a case where there are more than five streams, a waterfall can be used to show streams activity, and understand when anomalies exist.

Packetization tool shown in fig 4.13 uses different colors to show how the QUIC packets transmitted in the network are composed of QUIC and HTTP/3 frames. In the figure, the different colors are layered and have the following interpretation:

- black and grey on the bottom row are used to indicate the QUIC packets

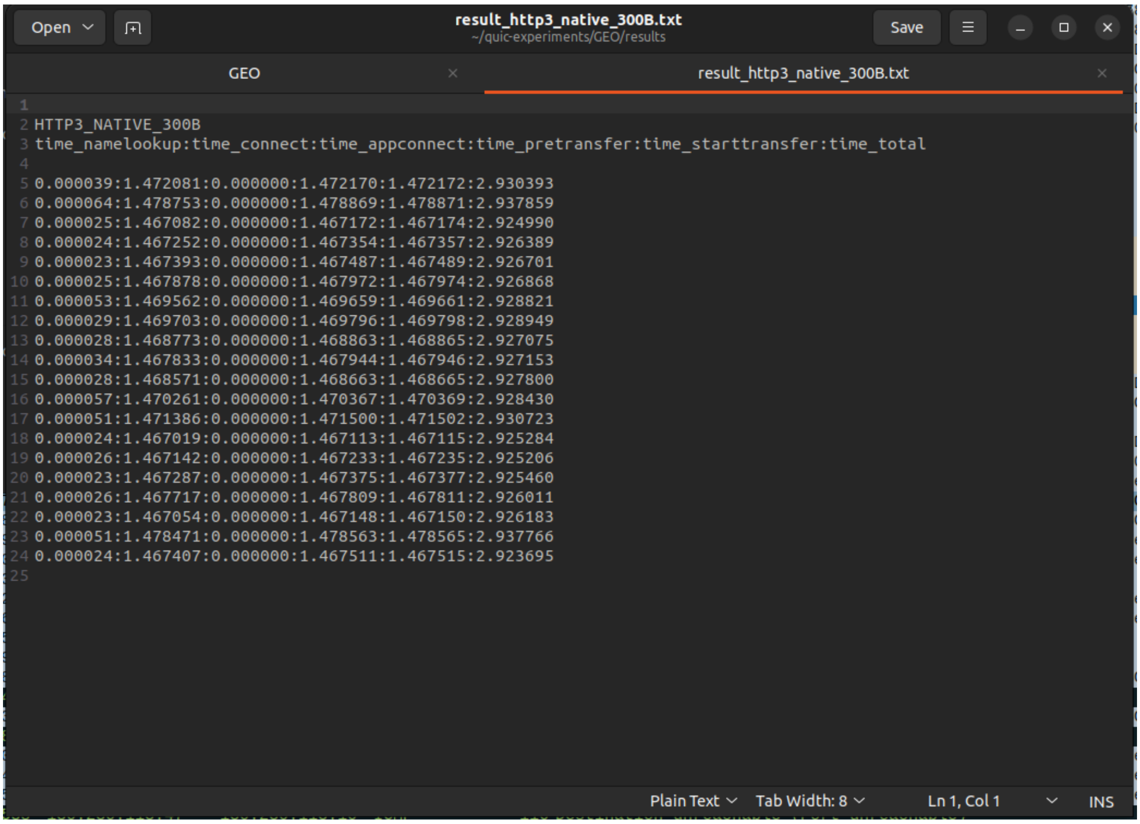

Figure 4.4. *Result of GEO Capture*

- the next row in red and pink colors indicate the QUIC frame in the packet payloads
- the next row would be in blue and light blue colors, which will show the HTTP/3 frames inisded the QUIC stream frame's payload
- the last row which can be in different colors, will indicate which HTTP/3 frame belongs to whch stream.

The Stream IDs, HTTP/3, QUIC frames, QUIC packets are plotted against bytes received or bytes sent.

In the sequence tool seen in fig 4.14 and fig 4.15, it is a diagram plotted vertically. This sequence tool gives a good illustration on timing, how the data flows between client and server. It begins with *connection started*, which is when the client has initiated the connection establishment with the server. The tool shows inactive period between the client and server, when there is a connection id change – an update occurs, the round trip time taken, acknowledgments (ack), and when the communication between the client and server ends, there is a *connection close*. This tool essentially makes use of logs of both client and server.

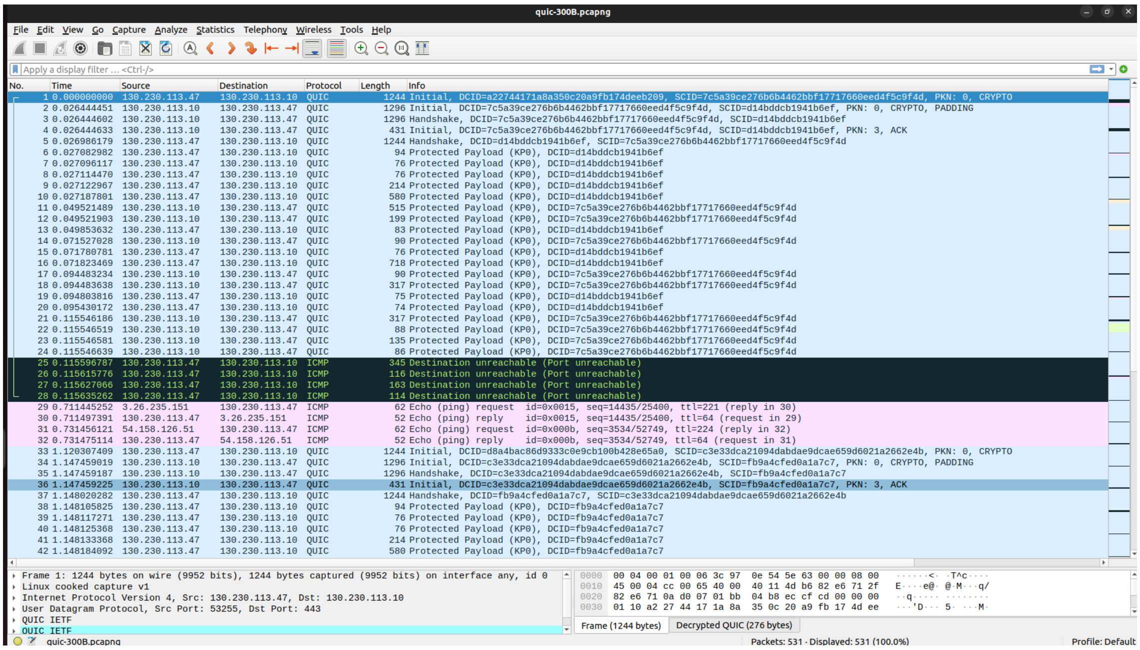

Figure 4.5. *LTE Wireshark Capture*

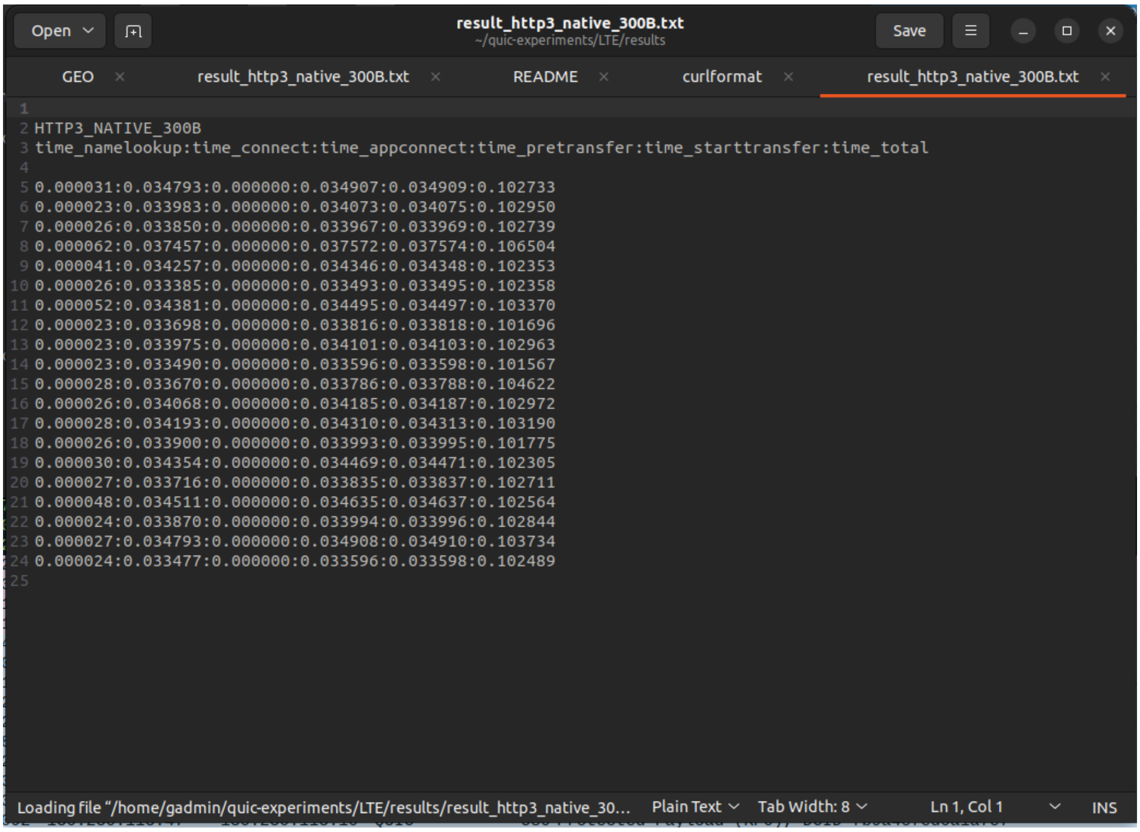

Figure 4.6. *Result of LTE Capture*

Figure 4.7. *QUIC and HTTP/3 Visualization Toolsuite*

```json
{
  "qlog_version": "draft-02",
  "qlog_format": "JSON",
  "title": "",
  "description": "",
  "summary": {},
  "traces": [
    {
      "vantage_point": {
        "name": "TODO",
        "type": "network",
        "flow": "client"
      },
      "title": "Connection 1",
      "description": "",
      "configuration": {
        "time_offset": "0",
        "time_units": "ms",
        "original_uris": [
          "file:///srv/pcap2qlog/examples/draft-01/spin_bit.json"
        ]
      },
      "common_fields": {
        "group_id": "1b51237b269288d6",
        "protocol_type": "QUIC",
        "reference_time": "1564682471.651907",
        "time_format": "relative"
      },
      "events": [
        {
          "time": "0",
          "name": "connectivity:connection_started",
          "data": {
            "ip_version": "4",
            "src_ip": "66.70.231.124",
            "dst_ip": "51.15.3.76",
            "transport_protocol": "UDP",
            "src_port": "52740",
            "dst_port": "4433",
            "quic_version": "0xff000016",
            "src_cid": "1b51237b269288d6",
            "dst_cid": "0f721e1c6aae0420",
            "trigger": "line"
          }
        },
        {
          "time": "0",
          "name": "transport:packet_sent",
          "data": {
            "header": {
              "version": "0xff000016",
              "scid": "1b51237b269288d6",
              "dcid": "0f721e1c6aae0420",
              "scil": "8",
              "dcil": "8",
              "packet_number": "0",
              "packet_type": "initial"
            },
            "frames": [
              {
                "frame_type": "crypto",
                "offset": "0",
                "length": "278"
              },
              {
                "frame_type": "padding"
              }
            ],
            "raw": {
              "length": 1251,
              "payload_length": 1224
            },
            "trigger": "line"
          }
        },
        {
          "time": "93",
          "name": "transport:packet_received",
          "data": {
```


Figure 4.8. *Sample Qlog Trace*

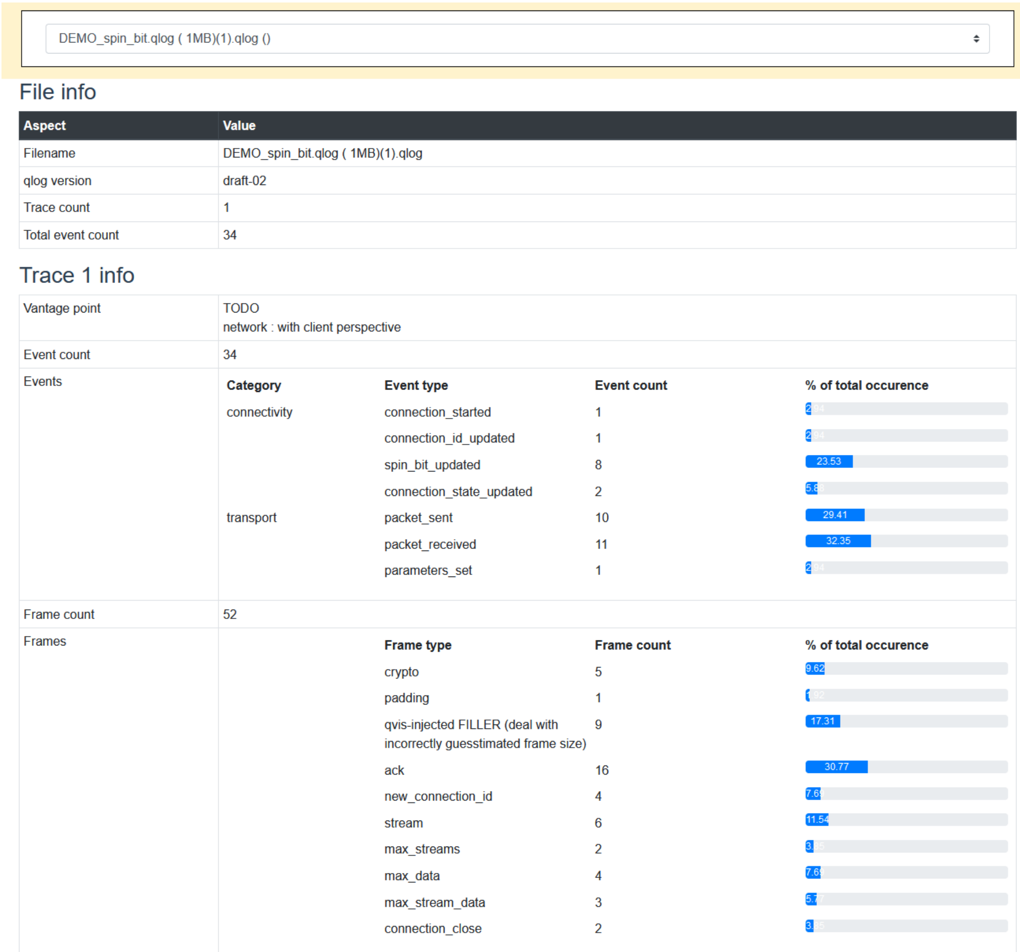

Figure 4.9. *Qlog Stats*

		spin_bit_updated	8	23.53
		connection_state_updated	2	5.8
	transport	packet_sent	10	29.41
		packet_received	11	32.35
		parameters_set	1	2.94
Frame count	52			
Frames		**Frame type**	**Frame count**	**% of total occurence**
		crypto	5	9.62
		padding	1	1.92
		qvis-injected FILLER (deal with incorrectly guesstimated frame size)	9	17.31
		ack	16	30.77
		new_connection_id	4	7.6
		stream	6	11.54
		max_streams	2	3.
		max_data	4	7.6
		max_stream_data	3	5.7
		connection_close	2	3.
Encryption level count	3			
Encryption levels		**Encryption level**	**Packet count**	
		initial	3	
		handshake	3	
		1RTT	15	
Connection-level Flow Control evolution (MAX_DATA, initial_max_data) Read as: viewpoint allows the other side to send this much data on the entire connection (all streams combined)		**Viewpoint**	**Evolution (bytes)**	
		Local (network)	32975 33389	
		Remote (client)	10000017 10000051	
Stream-level Flow Control evolution (MAX_STREAM_DATA, initial_max_stream_data_*) Read as: viewpoint allows the other side to send this much data on each individual stream	No stream level flow control limits set			

Figure 4.10. *Qlog Stats*

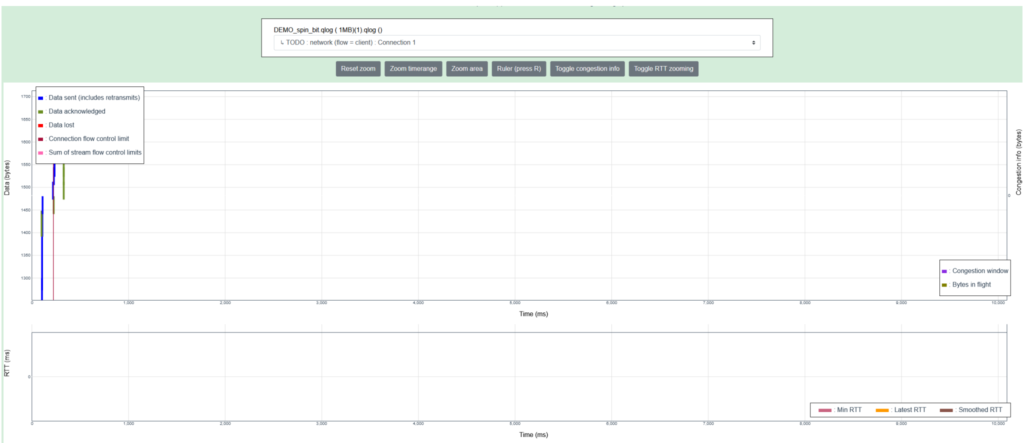

Figure 4.11. *QVIS - Congestion Tool*

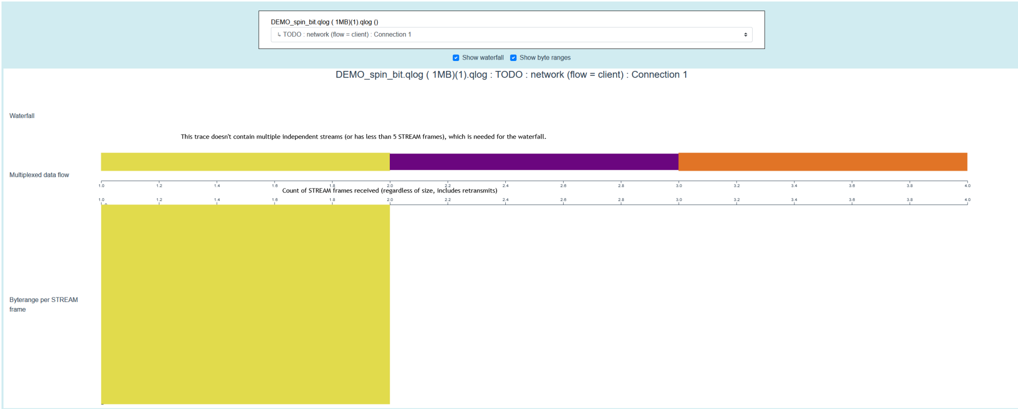

Figure 4.12. *QVIS - Multiplexing Tool*

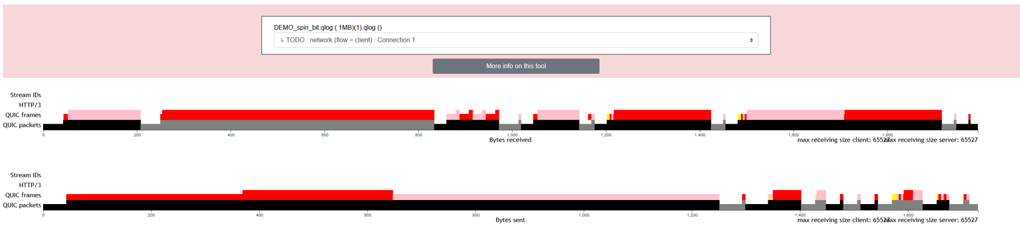

Figure 4.13. *QVIS - Packetization Tool*

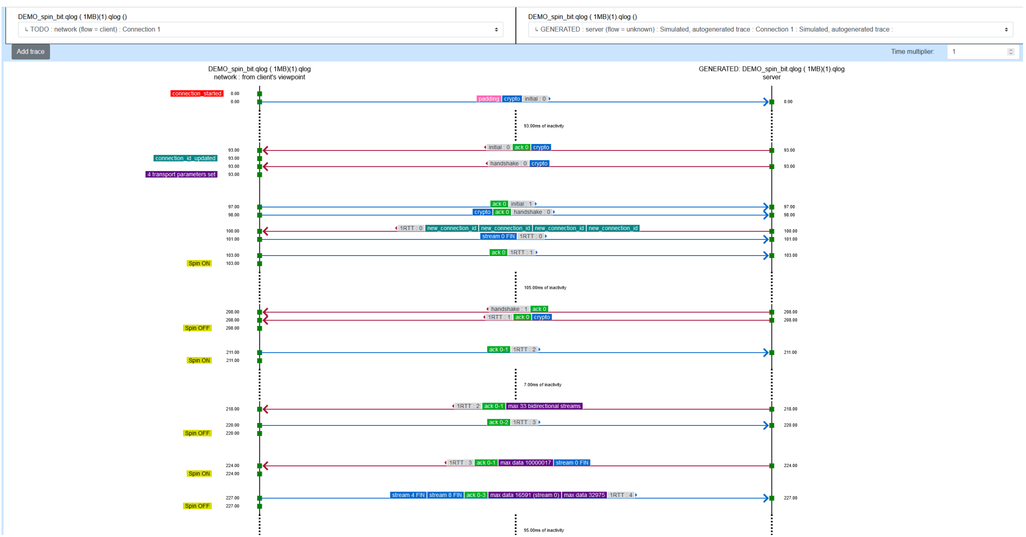

Figure 4.14. *QVIS - Sequence Tool*

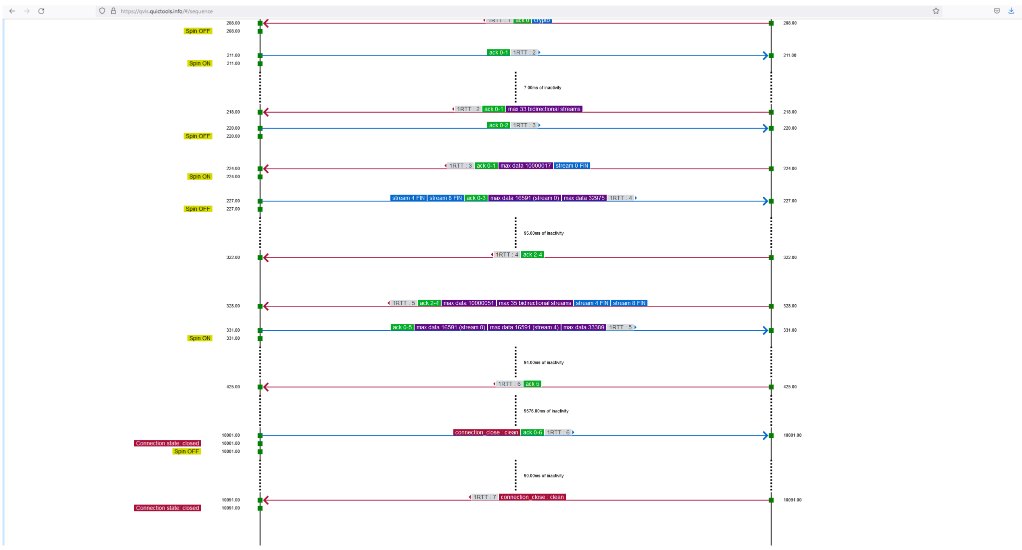

Figure 4.15. *QVIS - Sequence Tool*

5. CONCLUSION

This chapter will conclude the thesis work by highlighting and answering the research questions presented in the introduction part – chapter 1. To do this, previous chapter will be referenced and subsequent conclusions will be .

The first research question asked was "What is the current state of QUIC protocol?" This question was answered in 2.3 In this section, it was established that QUIC was founded in the last decade, but only standardized about two years ago, which makes QUIC a fairly new protocol. QUIC is getting accepted by large organizations, which is a positive sign towards its widespread.

The second research question of this thesis was "How does QUIC perform in networks, and the feasibility of the tools used in inspecting QUIC?" In section 2.4 the properties of QUIC give a detailed description of the performance of QUIC. One of the major reasons behind the development of QUIC was lower latency, due to emergence of latency-sensitive applications and high demands in bandwidth, which was unsustainable for HTTP. QUIC has been able to achieve lower latency with its one round trip time for first time connection, and zero round trip time for subsequent connections. This round trip time was shown in the wireshark capture in fig , With previous HTTP protocols, Head-of-Line blocking was a problem, which QUIC eliminates by allowing endpoints that are communicating to have multiple concurrent streams exist between them - multiplexing. In line with stream multiplexing, forward error correction helps to swiftly recover lost packets. Congestion control is an extension of stream multiplexing, which allows QUIC traffic congestion to be shared on streams provided from multiplexing.

One of the best properties of QUIC, which helps greatly in its performance is connection migration. In this digital age where people are on the move and still want to have a seamless Internet experience, connection migration will help connections remain alive, if there happens to be any change of port or client IP.

The security of QUIC is another property that enhances the performance of QUIC, and also answers the final question of the thesis "What security does QUIC protocol offer?" In section 2.4.1, it tells us that QUIC's security is built-in, it is an encrypted-by-default transport protocol, which encrypts its packet header and payload, to prevent modification. The handshake security provided by TLS v1.3 protects against handshake Denial of Service, but the 0-RTT connection establishment can introduce replay attack to the protocol. IP address spoofing, wire protocol, handshake costs and replay attacks were some of the security issues of QUIC, and how to address them discussed in section ...

The tools used in inspecting QUIC were described in chapter 3, and for most of it, they were very feasible. Wireshark is a a very popular protocol analyzer, which was used in capturing QUIC traffic. SSLKEYLOGFILE, curl, Ubuntu 22.04, web browser, were all easy to work with, it was just important to work with the most current version of tools to be used.

Chapter 4 showed some demos of captured traffic gotten from the github of qlog and qvis, to give insight to how qlog and qvis' statistics and visualization will pan out. The sequence, congestion, packetization and multiplexing tools were illustrated, along with qlog's logging format.

In the security aspect of QUIC, this thesis only described few of the attacks that QUIC can protect the average user from. It also explains how some of the properties which should be advantageous, actually introduce some of the security flaws of QUIC. There are still some attacks that QUIC is susceptible to, which QUIC has not being developed to protect users of QUIC from.

This thesis has provided a good amount of research into QUIC, by establishing its properties, exploring its flaws and security, the tools used in capturing and inspecting, and visualization. It shows that QUIC protocol is very promising, it will give its users better experience, and this thesis is open to further and deeper research into QUIC.

In this study, the challenge encountered became a limitation to this study. HTTP/3 and QUIC traffic captured with wireshark has the file format in *.pcap*, while qlog and qvis visualization tools do not support the *.pcap* format. This prevented the visualization of the captured traffic during the experiment, in the result stage. Wireshark is a popular tool used for network packet analysis, it will be important and necessary for its file format to be supported by qlog and qvis tools in the nearest future.

REFERENCES

[1] Accreditly. *The Differences Between HTTP 1.1, HTTP/2 and HTTP/3*. Accessed on 23-Jun-2023. 19 04 2023. URL: https://accreditly.io/articles/the-differences-between-http-11-http2-and-http3.

[2] Saleh Alawaji. *IETF QUIC v1 Design*. Accessed on 22-Aug-2023. 2021. URL: https://www.cse.wustl.edu/~jain/cse570-21/ftp/quic.pdf.

[3] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. "HTTP over UDP: an Experimental Investigation of QUIC". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015, pp. 609–614.

[4] Efstratios Chatzoglou et al. "Revisiting QUIC attacks: A comprehensive review on QUIC security and a hands-on study". In: *International Journal of Information Security* (2022), pp. 1–19.

[5] Comptia. *Comptia.org*. URL: https://www.comptia.org/content/guides/what-is-a-network-protocol.

[6] Sarah Cook et al. "QUIC: Better for what and for whom?" In: *2017 IEEE International Conference on Communications (ICC)*. 2017, pp. 1–6. DOI: 10.1109/ICC.2017.7997281.

[7] Yong Cui et al. "Innovating transport with QUIC: Design approaches and research challenges". In: *IEEE Internet Computing* 21.2 (2017), pp. 72–76.

[8] Bruno Volpato Da Cunha et al. "Performance Benchmarking of the QUIC Transport Protocol". eng. In: *Proceedings - IEEE Consumer Communications and Networking Conference, CCNC*. Vol. 2023-. 2023, pp. 206–212. ISBN: 9781665497343.

[9] Jacobian Engineering. *Jacobian Engineering*. Accessed on 22-Aug-2023. 2021. URL: https://jacobianengineering.com/blog/2016/11/1543/#:~:text=IP%20Address%20Spoofing,token%20to%20the%20client%20IP.

[10] Alexander R Galloway. "Protocol". In: *Theory, Culture & Society* 23.2-3 (2006), pp. 317–320.

[11] Alessandro Ghedini. *The Road to QUIC*. 28-Jun-2023. 2018. URL: https://blog.cloudflare.com/http-3-from-root-to-tip/.

[12] Geoff Huston. *A quick look at QUIC*. Accessed on 10-Jul-2023. 2019. URL: https://blog.apnic.net/2019/03/04/a-quick-look-at-quic/.

[13] Dominik von Künßberg and Benedikt Jaeger. "Debugging QUIC and HTTP/3 with qlog and qvis". In: *Network* 25 (2021).

[14] Ulf Lamping and Ed Warnicke. "Wireshark user's guide". In: *Interface* 4.6 (2004), p. 1.

[15] Adam Langley et al. “The quic transport protocol: Design and internet-scale deployment”. In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196.

[16] Robert Lychev et al. “How secure and quick is QUIC? Provable security and performance analyses”. In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 214–231.

[17] Robin Marx, Wim Lamotte, and Peter Quax. “Visualizing QUIC and HTTP/3 with qlog and qvis”. In: *Proceedings of the SIGCOMM’20 Poster and Demo Sessions*. 2020, pp. 42–43.

[18] Robin Marx et al. “Debugging QUIC and HTTP/3 with qlog and qvis”. In: *Proceedings of the Applied Networking Research Workshop*. 2020, pp. 58–66.

[19] Robin Marx et al. “Debugging QUIC with qlog and QUICvis”. In: (2020). Accessed on 22-Jul-2023.

[20] Robin Marx et al. *Main logging schema for qlog*. Internet-Draft draft-ietf-quic-qlog-main-schema-06. Work in Progress. Internet Engineering Task Force, July 2023. 50 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-main-schema/06/.

[21] Mohamed Moulay, Fernando Díez Muñoz, and Vincenzo Mancuso. “On the Experimental Assessment of QUIC and Congestion Control Schemes in Cellular Networks”. In: *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*. 2021, pp. 1–8. DOI: 10.1109/MedComNet52149.2021.9501271.

[22] Késsia Nepomuceno et al. “QUIC and TCP: A Performance Evaluation”. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. 2018, pp. 00045–00051. DOI: 10.1109/ISCC.2018.8538687.

[23] B. Nithya et al. “Performance analysis of pluggable congestion control in QUIC protocol”. eng. In: *AIP Conference Proceedings*. Vol. 2424. 1. Melville: American Institute of Physics, 2022.

[24] Tom Decker Robin Marx Alex Yu and Jeremy Lainé. *qvis*. Accessed on 10-Oct-2023. 2019. URL: https://github.com/quiclog/qvis.

[25] Rakesh Seal. *Looking Into QUIC Packets in your Network*. 06 30, 2023. 2021. URL: https://www.keysight.com/blogs/tech/nwvs/2021/07/17/looking-into-quic-packets-in-your-network.

[26] Martin Thomson. *Version-Independent Properties of QUIC*. RFC 8999. May 2021. DOI: 10.17487/RFC8999. URL: https://www.rfc-editor.org/info/rfc8999.

[27] P. Vaderna, E. Stromberg, and T. Elteto. “Modelling performance of HTTP/1.1”. In: *GLOBECOM ’03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)*. Vol. 7. 2003, 3969–3973 vol.7. DOI: 10.1109/GLOCOM.2003.1258974.

[28] *Wireshark*. Accessed on 10-Oct-2023. URL: https://www.wireshark.org/news/.

[29] John Yannakopoulos. “Hypertext transfer protocol: A short course”. In: *University of Crete. August* (2003).