Abstract

DNS is one of the fundamental protocols which are widely have been used on the Internet for the purpose of the conversion from domain names to IP addresses. Privacy and data protection are not the only crucial elements while using the Internet, but efficiency is becoming more crucial. DNS has a various privacy protocols. The efficiency of these DNS privacy protocols may get improved.

The QUIC protocol does provide privacy and data protection, and potentially may improve the efficiency of DNS privacy compared with other existing protocols.

Content

1. Introduction

1.1 Project Goal

The aim of this project is to complete the prototype of DNS/QUIC proof of concept test design and implementation, and time DNS/QUIC in comparison with the other various DNS privacy and plain DNS. Domain Name System (DNS) networking protocol is widely used on the internet. This protocol takes the domain name and converts it to an IP address. DNS does the mapping between domain names and IP addresses using DNS servers which could recursively ask other DNS server to get a response.

1.2 Motivation

Networking security is essential in modern computing. While DNS is resolving domain names to IP addresses, attackers could eavesdrop packets which could contain crucial data. There are existing transport layer security protocols which encrypt or add integrity to DNS. QUIC (Quick UDP Internet Connection) is a transport layer network protocol is designed to be faster and having several advantages over other transport layer security protocols in theory. This project investigates and discuss the implementation of DNS/QUIC.

1.3 Report Structure

The Background chapter will provide the reader with the required background information to understand the architecture of DNS and QUIC protocols.

Popular and existing DNS privacy protocols will be outlined in the Literature Review chapter. Advantages and disadvantages of each protocol is will be discussed.

The architecture of DNS/QUIC will be discussed in detail in the Design chapter.

A prototype application for DNS/QUIC protocol was developed. This prototype application will be explained in detail in the Implementation chapter.

The result and timing of running DNS/QUIC is will be discussed and explained in comparison with other existing transport layer security protocols.

Potential improvements and enhancement that could be added to DNS/QUIC prototype will be proposed in the Future Work chapter.

Analysis of the project will be discussed in the Conclusion chapter.

2. Background

There are few networking protocols and concepts will be referenced throughout this report. These protocols will be discussed and explain in this chapter.

2.1 Transport Layer

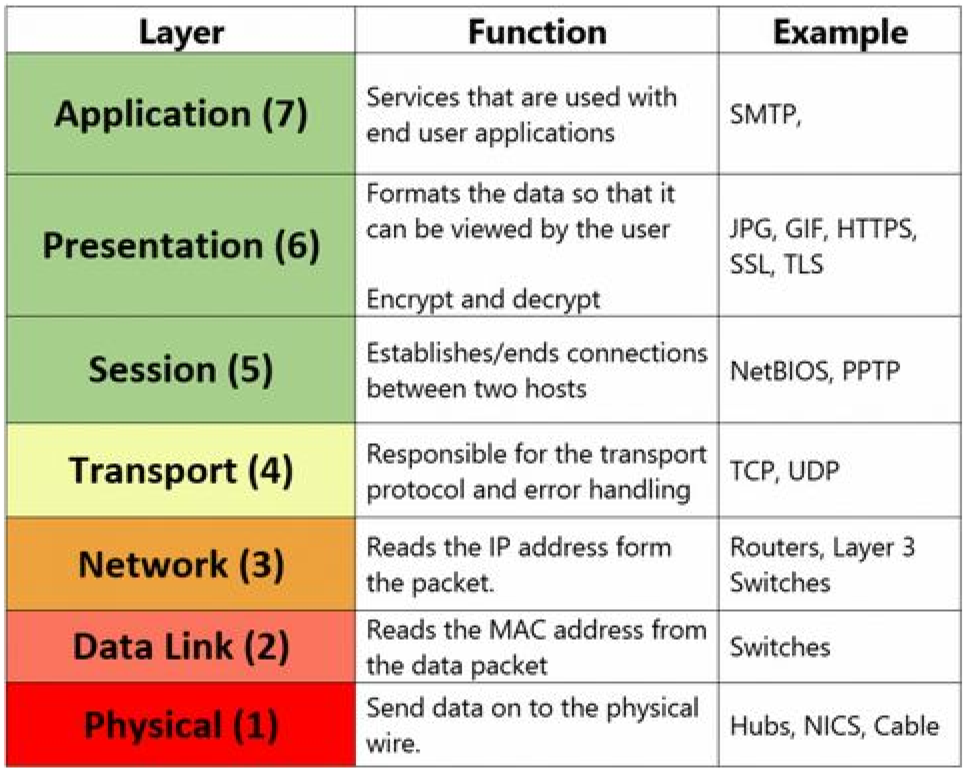
The transport layer provides end-to-end communications for applications. There are two protocols are primarily used in transport layer:

1. **User Datagram Protocol (UDP)**

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol [1]. UDP adds a minimum overhead to the connection. This is due to the fact that UDP does not need to make connection establishment or termination. UDP does not add any error checking or congestion control, which makes the connection faster but unreliable for transferring important data. For example, if a packet get lost, then UDP will not request a retransmission of the lost packet. There are several protocols use UDP and and add reliability at the application layer. UDP is used to send and receive a small packets is size. We conclude that UDP is used for establishing low latency connection.

2. **Transmission Control Protocol (TCP)**

The Transmission Control Protocol (TCP) is a connection oriented protocol [1]. TCP has flow, error and congestion control. All layers below TCP in the OSI stack are unreliable for end-to-end communication. IP discard packets while sending if an error occurred. TCP/IP is widely used, but there are other protocols that use TCP, but not IP such as FIle Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP).



Layer	Function	Example
Application (7)	Services that are used with end user applications	SMTP,
Presentation (6)	Formats the data so that it can be viewed by the user Encrypt and decrypt	JPG, GIF, HTTPS, SSL, TLS
Session (5)	Establishes/ends connections between two hosts	NetBIOS, PPTP
Transport (4)	Responsible for the transport protocol and error handling	TCP, UDP
Network (3)	Reads the IP address form the packet.	Routers, Layer 3 Switches
Data Link (2)	Reads the MAC address from the data packet	Switches
Physical (1)	Send data on to the physical wire.	Hubs, NICS, Cable

Figure 2.1: The OSI stack defines a networking framework to implement protocols in layers

TCP provides a connection oriented protocol to the application layer. It enables reliable communication on account of the fact that if a packet is lost or corrupted, then TCP is responsible for handling retransmission. It is regarded as a connection-oriented because all state transition are communicated to the two parties of the connection. TCP does not precisely know when to retransmit data, hence it uses acknowledgment packets for determining whether retransmission is required or not. In TCP, the connection must be established prior to transmission, unlike UDP which does not to establish connection prior to transmission. TCP is reliable, but it has a high latency connection due to its reliability on delivering packets.

2.2 Transport Layer Security (TLS)

Transport Layer Security (TLS) encrypts data that sent over the internet. It provides end-to-end privacy and data integrity between two communicating applications. TLS has a layer called “TLS Handshake Protocol” which will be discussed in detail in Chapter 3. TLS is used in the application layer to provide privacy. For example, it could be used to secure DNS. TLS is implemented on top of TCP due to its reliability. Reliable medium is required, because an error or data corruption occurring during TLS handshake will result in failure of establishing the connection; however, TLS has been implemented on top of UDP as well. There are situations in which UDP is preferable and TLS is used to secure the connection and it is called “DTLS” protocol.

2.3 Domain Name System (DNS)

The Domain Name System (DNS) is an application layer protocol which runs on top of UDP. It is responsible of converting alphabetic web addresses to IP addresses, which computers use to connect to the desired web server which has the requested web page.

Running DNS on top of UDP does not add data integrity or encryption to the packets and it is called “plain DNS”; however, DNS/DTLS protocol adds data integrity and encryption. Also, DNS could run on top of TCP and TLS to provide privacy and encryption and it is called “DNS/TLS”. These two protocols are already exist and will be discussed in Chapter 3.

2.3.1 Domain Name Space

The Internet is growing rapidly, and the number of users is increasing; hence, there are a huge number of web servers. Those servers are mapped using their name spaces to the corresponding IP addresses. Name spaces must be unique to be mapped correctly to the corresponding IP address to avoid ambiguity. Names are organised in a hierarchical structure.

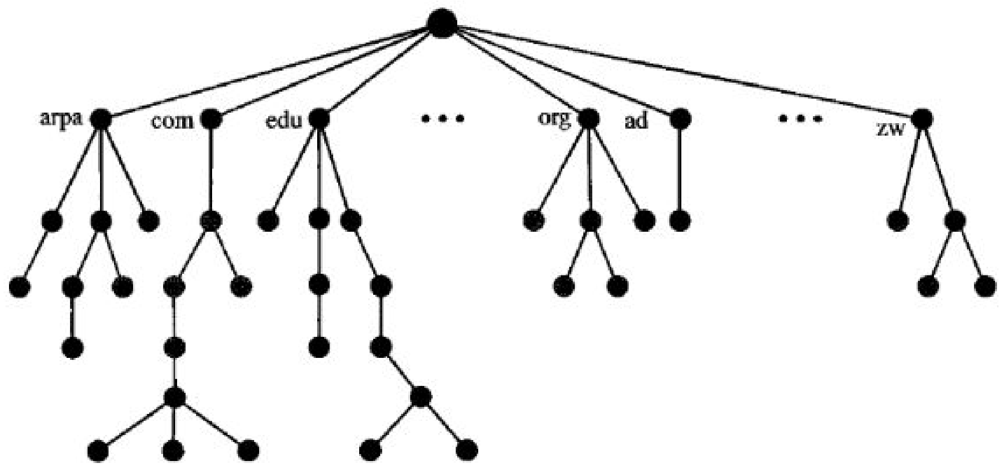


Figure 2.3.1: Domain Name Space hierarchical architecture

The hierarchical is an inverted binary tree. It has the maximum of 128 nodes. The first node on top of the tree is called “root”. Each node has a label. Parent nodes could have multiple children nodes. Children of a node have labels, but each label must be unique, which guarantees unambiguity of domain names. Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots(.)[1]. The next level of the tree is called Top Level Domains (TLD). This layer has .com, .org and etc.

Domains are a subtree of the domain name space, and the name of the domain is the name of the node on top of the subtree.

2.3.2 Resolution

Converting or mapping name to IP address is called “address resolution”. DNS is designed in client-server architecture. The host which request a mapping from a name to IP address is called a “resolver”. The resolver calls the closest DNS server, if that server could do the mapping, then it resolves the query and sends it back to the resolver. Otherwise, it asks other servers to resolve the query.

2.3.3 Recursive Resolution

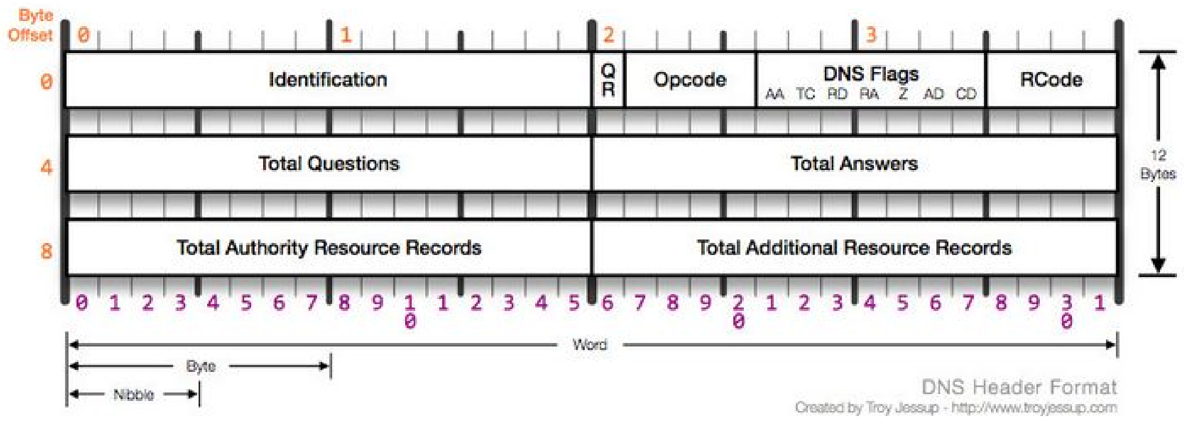
In DNS, there are few types of servers: stub, authoritative, and recursive. Stub servers usually initiate DNS queries. Authoritative servers store DNS record information. A resolver ask a DNS server to resolve name to IP address. If the server is an authority server of the domain name, then it search its database and returns an answer. Otherwise, it sends the request to another DNS server and waits for a response. This process is repeated until it reaches an authoritative server which finally replies with an answer. The response from the authoritative server travels back to the resolver. This process is called “recursive resolution”, and server which send DNS requests to another server searching for a response are called “Recursive servers”.

2.3.4 DNS Cache

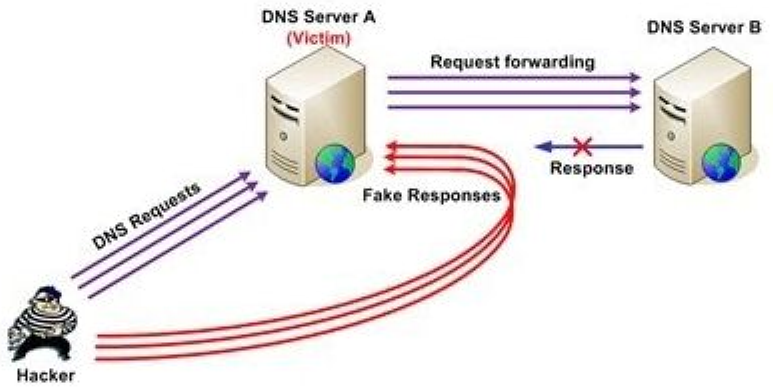
DNS has caching, which will cause a reduction in time for mapping names to IP addresses through recursive resolution. DNS resolver requests an answer for a query, and this response will be cached, so the resolver will not have to resolve names using recursive resolution, which will be more efficient. Authoritative servers always adds information called “time-to-live”. This information defines the time in seconds that the receiving server can cache the information[1]. After this time elapses, cached information will be expired and the query must be sent again to the authoritative server[1].

2.3.4 DNS Architecture

DNS has two different types of messages: query and response[1]. DNS query packet consist of header and question section; the response packet consist of header, answer record, authoritative record, and additional records. The header part in both query and response packet is identical.



The identification field is for the client to match the query and response identification number. Both query and response must have the same identification number to be accepted at the client. Otherwise, response will be dropped. This check is mainly for security reasons to prevent DNS cache spoofing.



DNS resolver could sent a query and an attacker could also eavesdrop the connection and send invalid or incorrect DNS record information to the resolver before it gets a response from an authoritative or recursive server. This information will be cached and everytime the resolver tries to resolve that name to IP address will get the incorrect mapping. This is called DNS cache spoofing. DNS header has identification field which randomly assigned to the DNS query, and the resolver will not accept any response that does not match its identification number in the query. This limits cache poisoning because attackers will have to guess this random identification number to be able to spoof the cache. The QR field identifies whether the message is query or response. The standard port to run DNS on is 53 on both UDP and TCP; however, other ports could be used.

2.4 Quick UDP Internet Connection (QUIC)

Quick UDP Internet Connection (QUIC) protocol is a secure transport layer protocol which runs over UDP. This protocol was designed to be secure and and flexible, and having several advantageous over other secure transport protocols such as TLS and DTLS. QUIC aims to have low-latency connection establishment, authenticated and encrypted payload, and stream multiplexing. QUIC uses TLS handshake to establish the connection. The specification of QUIC and the design of the protocol will be discussed in detail in Chapter 4.

3. Literature Review

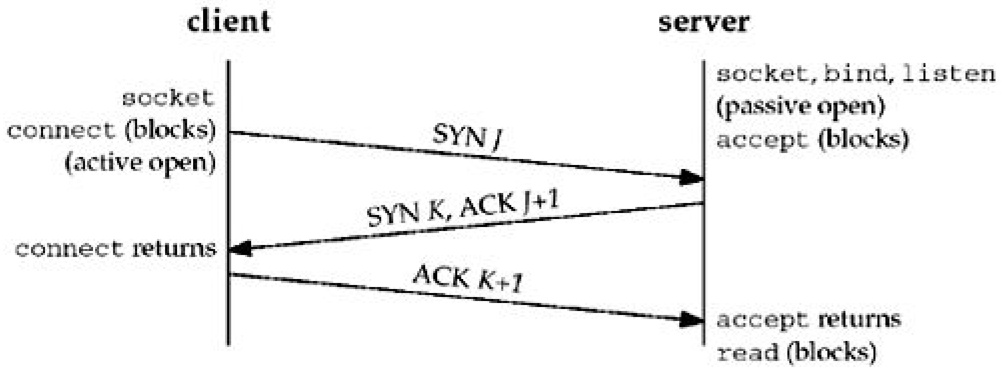
There are several existing protocols which provide secure connection for DNS. In this chapter, these protocols will be discussed in detail.

3.1 DNS/TLS

The standard is to run DNS over UDP, but it does not provide a secure connection. In order to secure DNS, TLS is used. Each step of running DNS/TLS will be discussed and explained.

3.1.1 TCP Session Establishment

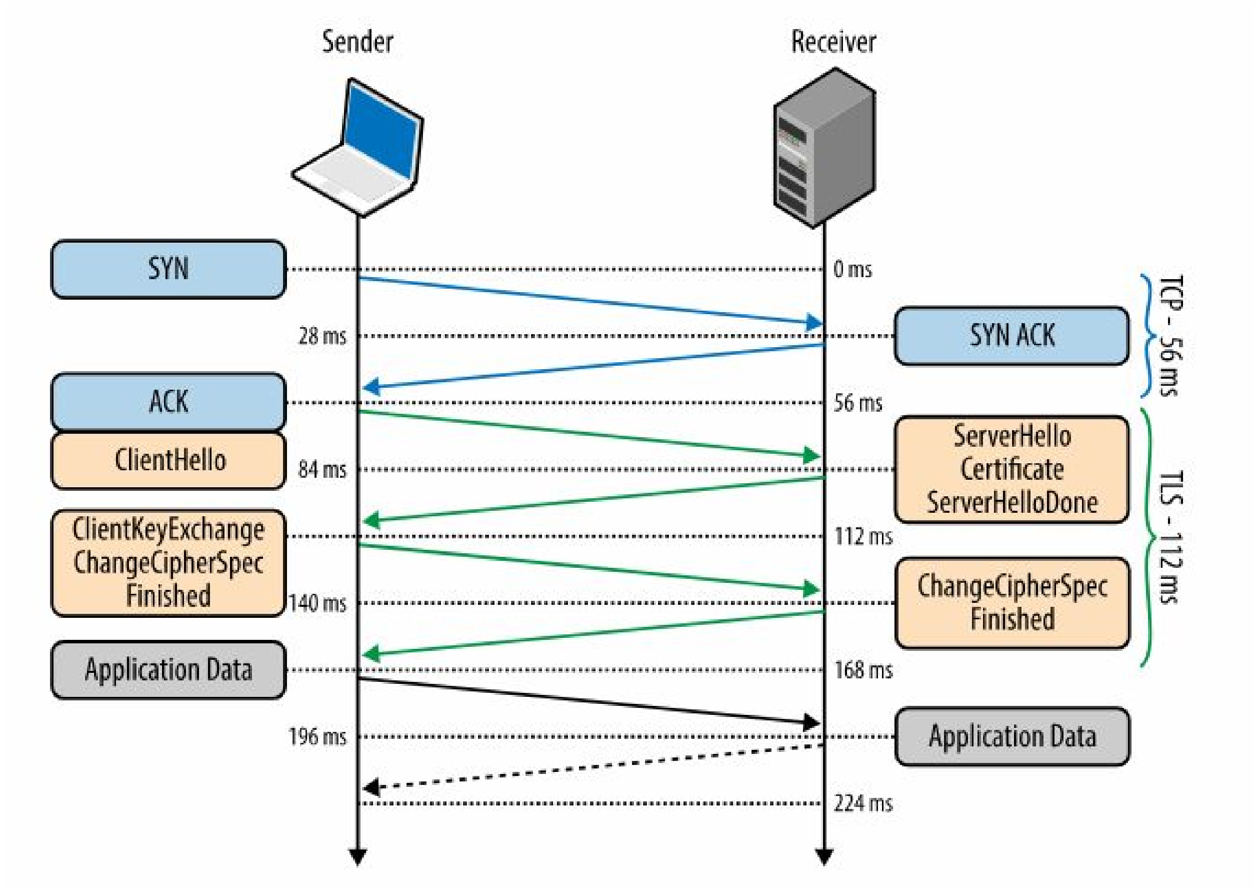
DNS server listens and establishes TCP connection on the designated port which is 853, unless it has an agreement with the client to use other port for running DNS-over-TLS [2].



TCP three-way handshake establishes TCP connection between client and server. The client sends synchronize (SYN) segment with client’s initial sequence number (denoted as “J” in the above diagram). Server listens to the client and replies with a synchronize (SYN) with the server’s initial sequence number (K), also an acknowledgement (ACK) is sent with the client’s initial sequence number incremented by 1 all in one segment. Then, the client replies with an acknowledgement (ACK) and the client’s initial sequence number incremented by 1. At the end of this process, the TCP connection is established. It is three-way handshake because each side of the connection need to verify that they can transmit and receive segments, unlike two-way handshake where only the client can send data and the server acknowledges it, but never get the chance to transmit data. Now, the TCP connection is established and ready for transmitting data.

3.1.2 TLS Handshake

The TLS handshake is used to authenticate client and server on their first communication. Once client and server success communicating via TCP, then they proceed with the TLS handshake [3]. The use of public and private key pair is the basis of the authentication. The public key is made accessible for the public. The private key is remain confidential for its owner. If a client and a server want to establish a secure connection and send encrypted data, then the client uses the server's public key to encrypt, and the server uses its private key to decrypt data.



Client sends in clear text the specification of the TLS protocol. It sends the version of TLS protocol it uses, and supported ciphersuites it may use. Then, the server sends its certificate which has the server's public key to the client for authentication. The client sends its private key encrypted with server's public key. Then, both client and server send finish segment, then both will be ready now for transmitting and receiving encrypted data.

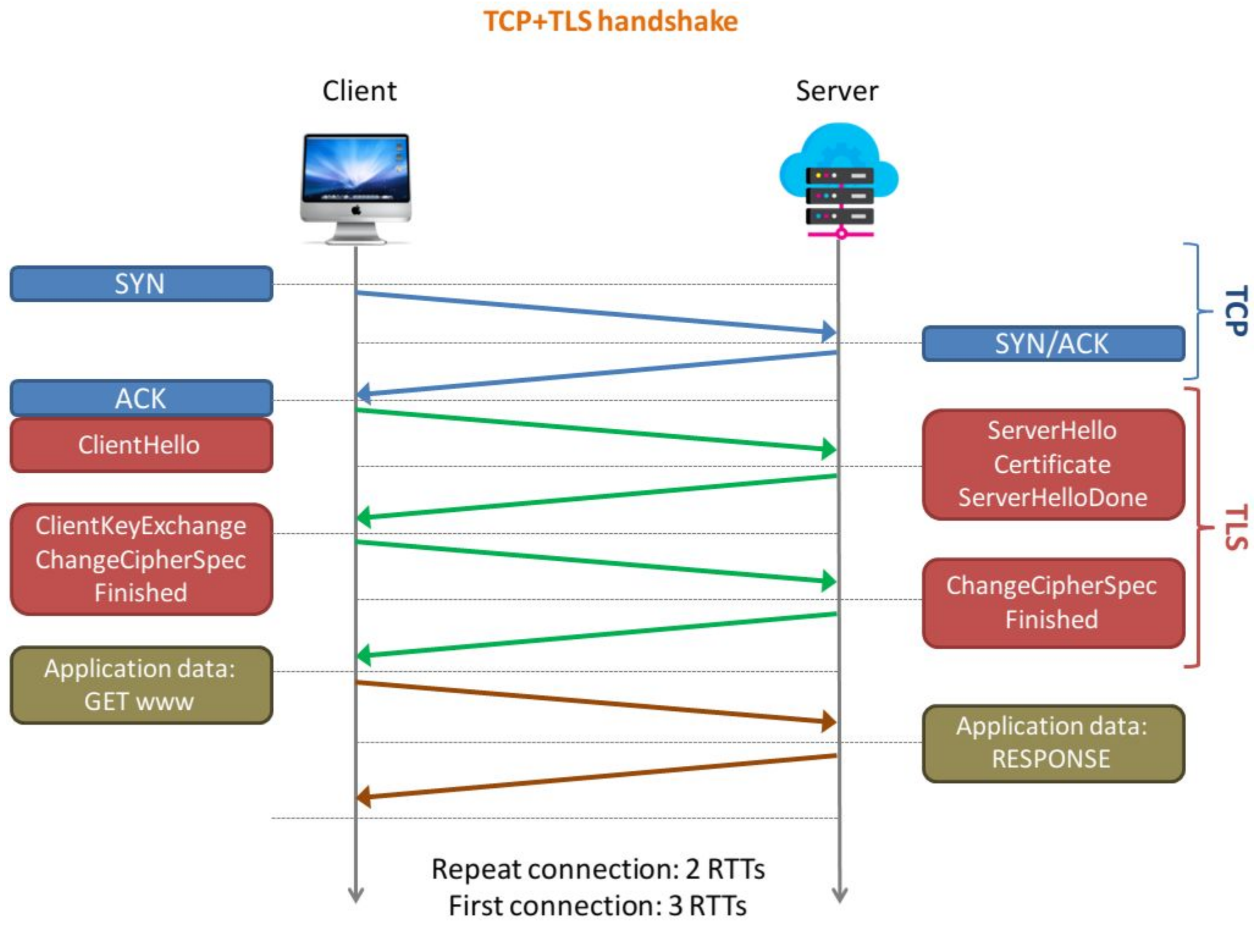
3.1.3 Transmitting and Receiving packets

Once the TCP connection and TLS handshake are established, then DNS client are server are ready for transmitting and receiving encrypted queries and responses. The

DNS client send a query to the DNS server which either has the associated IP address of the host name in the cache or the server will have to use recursion resolution to get a response.

3.1.4 Performance Concerns

Running DNS-over-TLS has few performance concerns such as latency and processing time. In this project, we are mainly concerned with the latency concerns.



Running DNS/TLS will add additional Round Trip Time (RTT) of latency for establishing TCP connection, and another two RTTs for the TLS handshake on the first communication between client and server (DNS client and server). In total, it takes 3 RTTs on first connection, and 2 RTTs on repeat connection, because TLS session resumption can only start after TCP connection is established. Plain DNS does not add any latency because it uses UDP; hence, there will be no need to establish TCP connection or making TLS handshake since plain DNS is not encrypted. TCP suffers from network head-of-line blocking, where the loss of a packet causes all other TCP segments to not be delivered to the application until the lost packet is retransmitted [4].

3.1.5 Security Concerns

(Do it later)

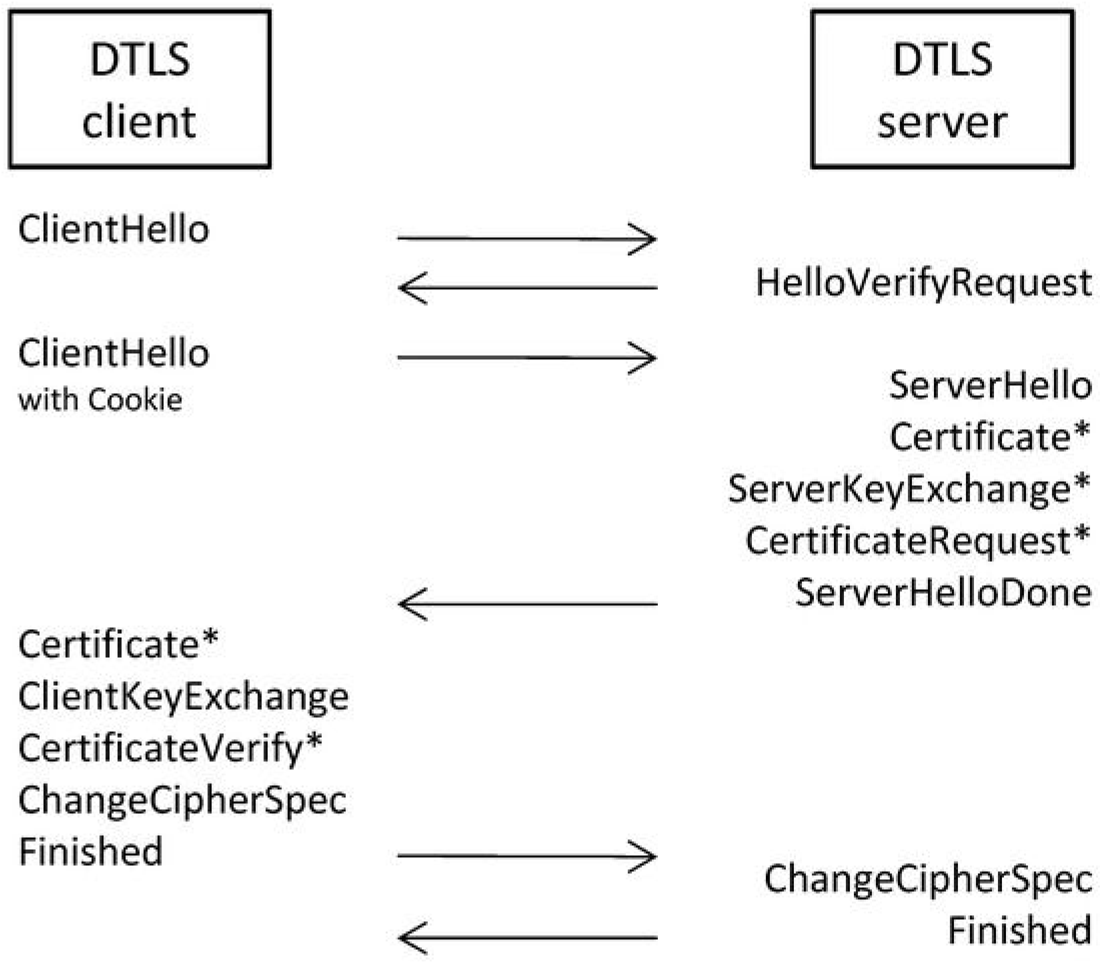
3.2 DNS/DTLS

Many application layer protocols prefer to use UDP due to its unreliability which makes the connection faster; therefore, DTLS was designed and implemented to provide privacy and integrity by encrypting datagrams of UDP. DTLS protocol uses TLS, but it modifies the original protocol. DTLS allows a retransmission of handshake messages to deal with the unreliability of UDP connection. DTLS does not suffer from network head-of-line blocking since it runs on top of UDP. Each step of running DNS/DTLS will be discussed and explained. DTLS message size is considered to be an issue. Usually TLS and DTLS handshake messages can be large and could go up to 2^24-1(many kilobytes). On the other hand, UDP datagrams are quite small. To compensate this problem and make the DTLS datagrams fit for the desired payload length as well, each DTLS handshake message contain both fragment offset and fragment length, then the recipient can reassemble all fragmented datagrams without using IP fragmentation which is very undesired because it adds an overhead [4].

3.2.1 Session Initiation

DNS-over-DTLS runs over the designated port, which it 853, unless client and server agree to use different port. DNS client checks if the DNS server supports DNS-over-DTLS by sending DTLS ClientHello to the server. If the server does not support DNS/DTLS, then it will not respond the DTLS ClientHello message. In that case, the client should retransmit DTLS Client hello, but it should stop after 15 seconds if the server does not respond.

3.2.2 DTLS Handshake



DTLS handels packet loss by allowing retransmission of handshake messages. This is done by adding a timer on ClientHello and ServerHello messages. The HelloVerifyRequest uses stateless cookie to handle and prevent denial of service (DoS) attack. The denial of service (Dos) is a cyber attack makes network resource unavailable to its users by flooding the victim server by sending malicious requests to overload it. In DTLS handshake, server send a stateless cookie, then the client receives it and has to send it back for authentication. This is a simulation of a three-way TCP handshake. After DTLS client and server authentication, ciphersuites and server's public key is exchanged, and this is similar to TLS handshake when private and public key pair is used for authentication.

3.2.3 Transmitting and Receiving packets

Once DTLS handshake is done successfully, then all data sent will be encrypted and attackers will not be able to eavesdrop packets or inject bogus DNS responses.

3.2.4 Performance Concerns

Running DNS-over-TLS suffers for the head-of-line blocking, unlike running DNS-over-DTLS because it runs on top on UDP. DTLS session resumption takes 2 RTT whereas TLS session resumption can only start after TCP handshake is complete [4]. Now, we can revisit the DTLS messages size and what potential performance issues does it add on DNS/DTLS. In comparison to plain DNS, running DNS-over-DTLS adds at least 13 octets (the octets is a telecommunication unit consist of 8 bits) of header, cipher and authentication overhead to every query and response. This will reduce the size of DNS payload that can be carried [4]. DNS client must inform the DNS server the maximum DNS response size it can re-assemble and deliver to the DNS client's network stack [4]. The server must consider the amount of expansion expected by the DTLS processing when calculating the size of DNS response response that fits within the path MTU [4]. If the DNS response exceeds the pre-calculated length that can fit within DTLS datagram (path MTU), then the DNS server informs the client, then the DNS client must a new DNS request over an encrypted transport such as DNS-over-TLS, which adds one or two RTTS of latency [4].

3.2.5 Security Concerns

One of the main security concerns that involved in running DNS/DTLS is the downgrade attack. As described above in the performance consideration section, sometimes running DNS/DTLS cannot be done due to the message size, and DNS/TLS must be used. In the worst case scenario where client or server do not support DNS/TLS, then they may fallback to use clear text to get a response. DNS clients keep track of servers that known to support DTLS to avoid downgrade attacks.

4. QUIC Protocol Design & DNS/QUIC Proof of Concept Test Design

4.1 Overview

In this chapter, the design of QUIC protocol design stages will be discussed in detail, then DNS-over-QUIC proof of concept test design is discussed.

4.2 QUIC Protocol Design

Quick UDP Internet Connection (QUIC) is a multiplexed and secure transport protocol which runs on top of UDP. QUIC protocol aims to flexible set of features that allow it to be a general-purpose secure transport for multiple applications [5]. QUIC has a set of features which are considered to be advantageous over other secure transport protocols. QUIC aims to provide low-latency connection establishment, authenticated and encrypted header and payload, stream and connection-level flow control, and stream multiplexing [5]. QUIC provides the same DNS privacy protection that DNS/TLS provides. QUIC is specifically designed to reduce latency by supporting 0-RTT data during session resumption, also it prevents head-of-line blocking by allowing parallel delivery on multiple streams of data using stream multiplexing technique. Currently, the only mapping defined with an application is HTTP-over-QUIC.

4.2.1 Using Transport Layer Security (TLS) to Secure QUIC

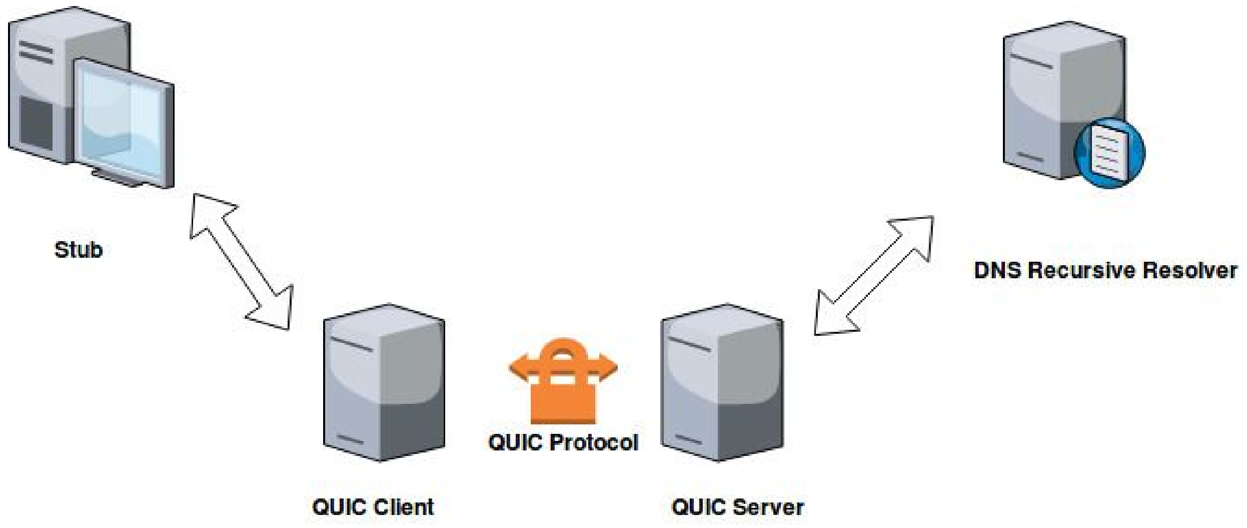
QUIC is secured using TLS version 1.3. TLS version 1.3 has various advantages from previous versions. It provides latency improvement for connection establishment, and first time connections can be established within 1 round trip (RTT); moreover, on subsequent connections between the same client and server, the client can transmit data immediately using 0-RTT feature.

4.2.1.1 TLS Handshake

TLS version 1.3 provides two basic handshakes that provides QUIC with latency improvement for connection establishment. Full 1-RTT handshake which the client is able to send application data to server after one round trip (1-RTT). The second handshake is 0-RTT handshake, the client uses information that has already learned about the server to send application data immediately with no latency. Zero RTT is only available if the client and server have previously communicated [6]. QUIC has multiple streams to transmit and receive data simultaneously and prevent the head-of-line blocking. QUIC reserves and uses stream 0 for TLS connection. QUIC permits client to send frames to streams, and the initial packet from client is a stream frame for stream 0 which contains the first TLS handshake message.

4.3 DNS/QUIC proof of Concept Test Design

There are different scenarios for using DNS protocol, and they can be classified in three different groups: stub to recursive resolver, recursive resolver to authoritative server, and server to server [7]. The design of DNS/QUIC only focuses on the stub to recursive resolver scenario.



Stub sends the DNS query to QUIC client. Data sent between QUIC client and server and encrypted using QUIC protocol; then, QUIC server receives DNS query from the client and forwards it to the DNS recursive resolver which looks for a response by communicating other DNS servers. In the current protocol design, DNS stub and recursive servers run on a different process than QUIC servers.

5. DNS/QUIC Proof of Concept Test Implementation

5.1 Implementation Overview

DNS resolver and QUIC library used in the project are called “Unbound” and “Picoquic” respectively. The protocol was designed to receive a DNS query at the stub in the DNS resolver initiated from the operating system and to be sent to the recursive server through QUIC servers which sit in the middle between the stub and recursive. In this implementation, QUIC library was modified to accept DNS queries and responses as it was originally designed to only accept HTTP requests.

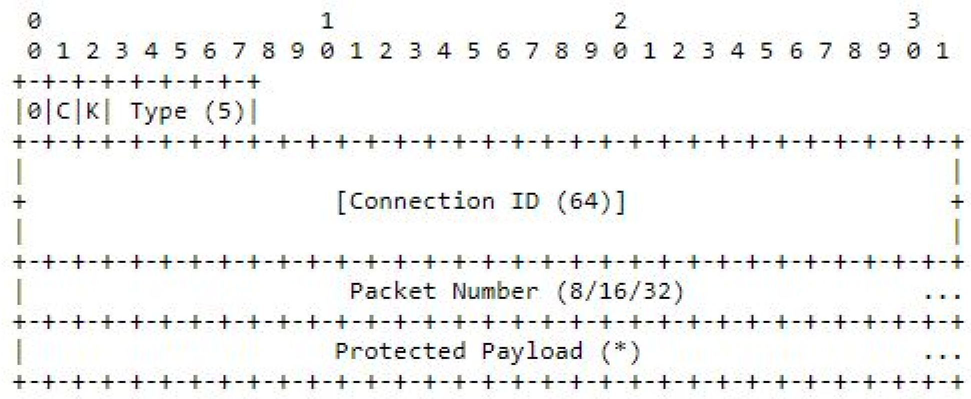
5.2 UDP Tunneling

On tunnel endpoint, UDP tunneling encapsulates the packet of DNS inside UDP datagram and transmit it to another tunnel endpoint, which decapsulates the UDP datagram and forwards the original packet contained in the payload to a different network [8]. UDP tunneling is used when middleboxes do not support payload protocol that may exist along the path of datagrams [8]. UDP tunneling is used to transmit packets carrying DNS queries and responses between stub and recursive servers through QUIC servers, because they do not support DNS protocol.

5.3 Encoding & Encrypting DNS Queries in QUIC

QUIC library was modified to run both end-point servers to complete the handshake; then, wait for incoming packets which will be carrying DNS queries. Domain name in DNS query must contain “www”. This part must be appended to the domain name that the query is carrying. Although DNS does not use World Wide Web protocol for resolution and it is not necessary to run DNS, but it was added in the implementation to eliminate a future DNS query corruption caused by QUIC servers. This issue will be discussed in the next section. Streams in QUIC protocol are designed to contain data that are desired to be sent encrypted. Stream 0 is used exclusively for sending the handshake packets. When QUIC client receives a packet carrying DNS query through UDP tunneling, it adds it to one of the available streams; then, all data within that stream will be encrypted and transmitted to the other QUIC endpoint.

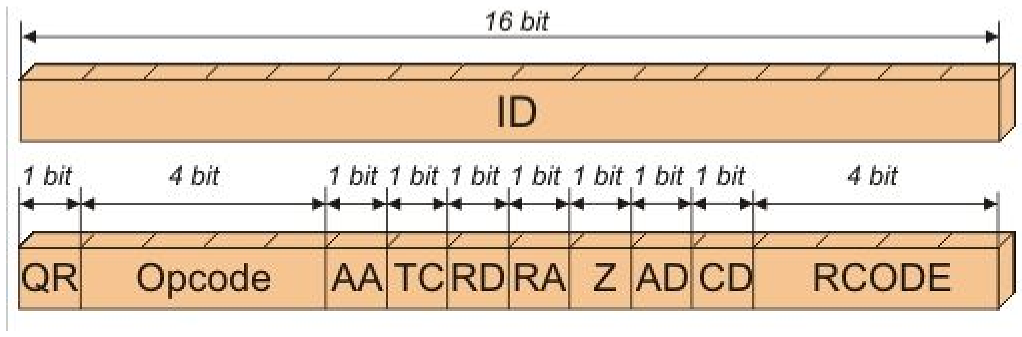
5.4 Decrypting DNS Queries and Responses in QUIC



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|0|C|K| Type (5)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                     [Connection ID (64)]                      +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Packet Number (8/16/32)                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Protected Payload (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

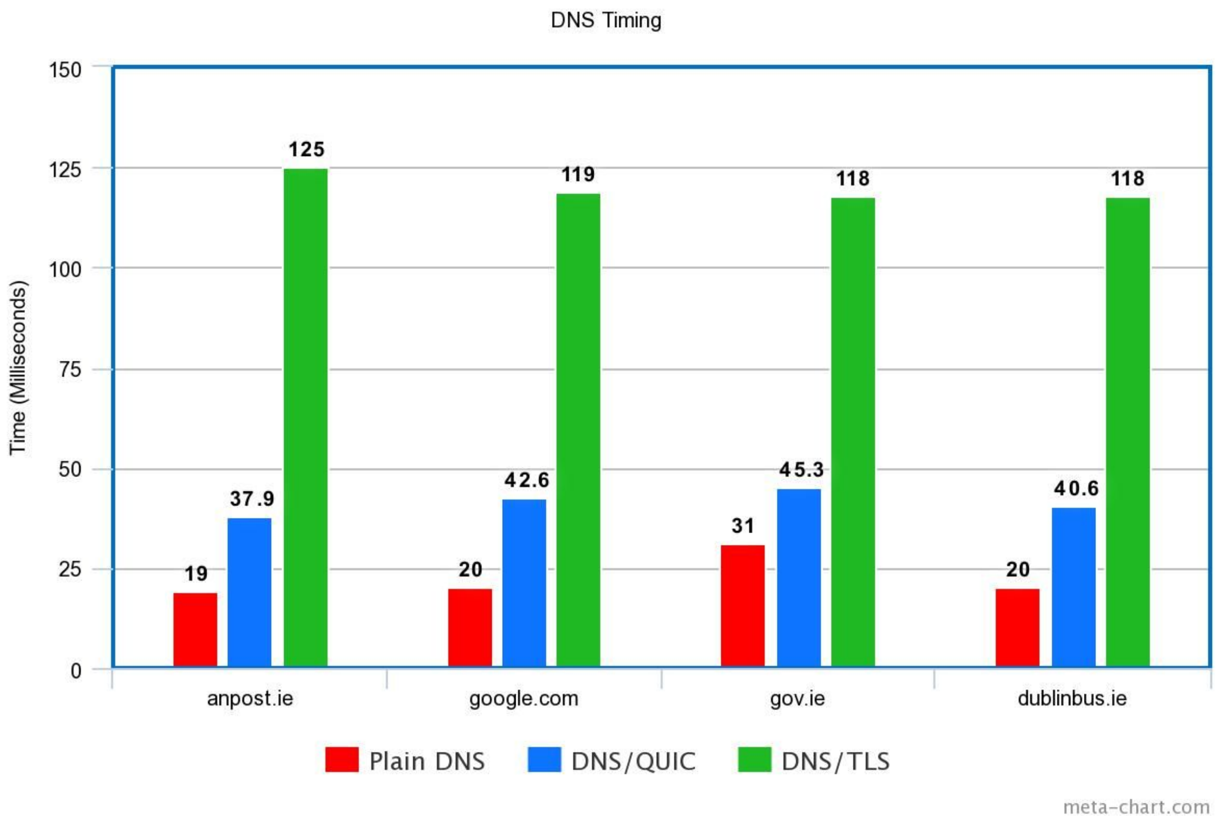
QUIC has two different types of headers are used in packets. Any QUIC packet either has a long or short header. Long headers are used in the handshake while exchanging keys and negotiation. Short headers are used after connection established is successfully done. Short headers are used to transmit protected payload.

Packet decapsulation has to be done to extract the protected payload. Packet decapsulation has to be done before transmitting its data to DNS servers. Otherwise, DNS servers will mark the DNS packet as malformed since they do not recognise it. After testing, it appeared that the length the header and data that are on top on the protected payload are dynamic and could change depending on the length of protected payload.



For packet decapsulation, the beginning of the DNS header in the protected payload has to be located in the received packet. The DNS header starts with the transaction ID, but it is randomly allocated to each DNS message, hence it cannot be used to find the beginning of the DNS header. An alternative solution would be search for the hexadecimal value of DNS flags which indicates whether the message is a query or a response. The hexadecimal value in DNS flag for indicating a message is a query or a response is 0x0120 and 0x8180 respectively. This approach was tested and did work, but in some irregular cases, either the hexadecimal value of 0x0120 or 0x8180 is happened to be part of the data that QUIC header that sits on top of the protected payload. In this irregular cases, DNS server will mark the DNS message as malformed and resolution will fail. To prevent and eliminate this issue, “www” has to be appended to the domain name when initiating the query. For example, the request from the operating system should look like “dig www.google.com”. This approach adds robustness and prevent DNS messages to be marked as malformed in the irregular cases. Using this approach, the beginning of the DNS header can be found without causing DNS server resolution to fail. The hexadecimal value for a query or response and the “www” substring in the domain name that is contained in the queries section on the DNS header have to be found, then the length of the DNS header is computed correctly and the rest of QUIC header is fully removed (packet decapsulation).

6. Results



In results, the timings were taken with empty stub and recursive cache. The ideal scenario was to get timings with empty stub, but full recursive cache to time transmission of packets between the stub and recursive only without looking at communication time between the recursive and other DNS or authoritative servers. In the DNS resolver library, both stub and recursive share the same cache, hence decided to clear the cache and include communication time between recursive and other servers to get responses. All tests are done in Trinity College Dublin. The recursive server forwards queries to TCD server. TCD server has a warm cache (full cache) which contain domain names resolution in cache.. The timings in the graph are the average of 100 runs for each domain name per protocol.

Running plain DNS is the fastest. This due to the fact that plain DNS runs over UDP which does not need any Round Trip TIme to connection establishment. Furthermore, plain DNS is transmitting DNS messages in plain text which means there is not any overhead added to resolution time to make handshake. DNS messages will be transmitted and a response will be sent back within 1-RTT.

Running DNS/QUIC is slower than plain DNS but faster than DNS/TLS. In the DNS/QUIC proof of concept test implementation, connection establishments always take 1-RTT and 0-RTT was not implemented. This means connection establishment which includes the handshake between the two QUIC endpoints takes 1-RTT before any application data could be transmitted. On average, QUIC handshake takes 15 milliseconds. For a adequate explanation, we can look at the domain name “anpost.ie”. Plain DNS transmits a query and receives back a response in 19 milliseconds. For DNS/QUIC, before a DNS messages can be exchanged , connection establishment begins with QUIC handshake which was timed and takes 15 milliseconds on average. Then, the DNS resolution starts which originally took 19 milliseconds in plain DNS. Adding up DNS resolution time “19 milliseconds” and QUIC connection establishment which is based on the handshake timing “15 milliseconds”, they add up to 34 milliseconds and that is close enough to DNS/QUIC timing for anpost.ie domain name which is “38 milliseconds”. The difference between the two timings could be explained as in the first possible scenario that QUIC handshake may took longer that 15 milliseconds or that QUIC servers added up latency while encoding, encrypting, and decrypting DNS messages.

Running DNS/TLS,

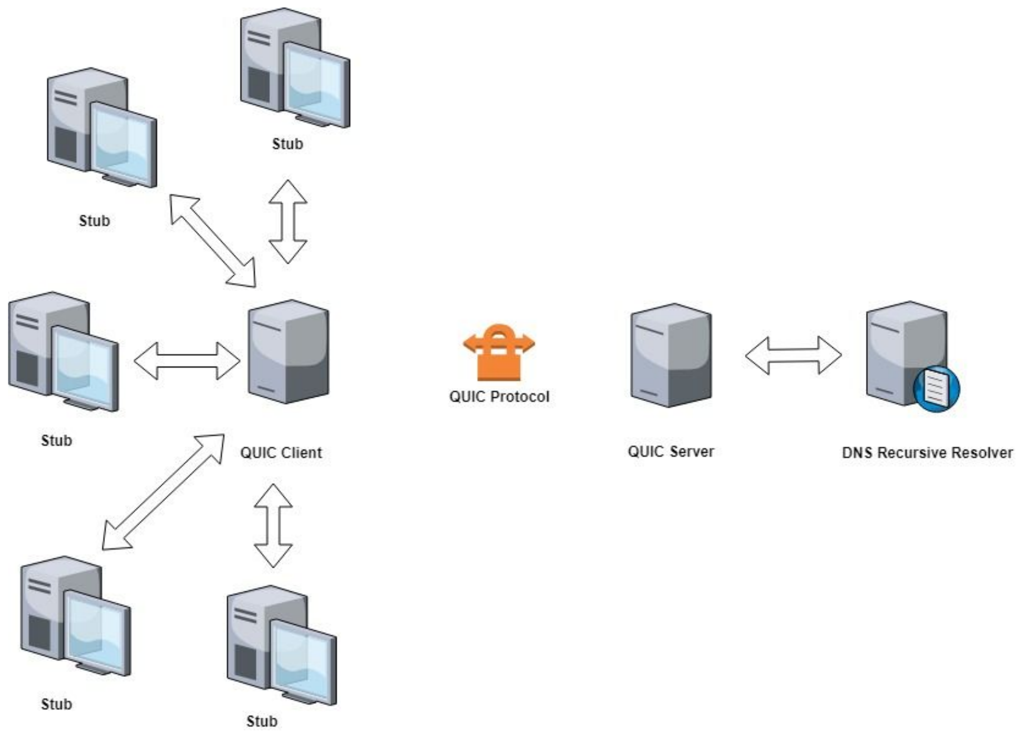
7. Future Work

7.1 Make QUIC Library Multithreaded

QUIC library which is used in this project “Picoquic” is a single threaded library. Running both of QUIC endpoints means that they run on a different processes. Converting the library to be multithreaded will make both QUIC endpoint run on the same process, hence it could resolve DNS queries faster, and it is essential to link DNS resolver library with QUIC library.

7.2 Link DNS Resolver Library with QUIC

DNS resolver “Unbound” is a multithreaded library. To link both libraries, QUIC library must be multithreaded. QUIC servers could be launched when DNS resolver library send a DNS query. Linking the two libraries could be done by running all of them on the same process which could reduce latency and make DNS resolution faster.



Multiplexing without the head-of-line blocking is one of the main advantages of QUIC over TLS. Due to the limitation of QUIC library used in this project, this feature was not investigated since the current proof of concept test design only accepts one stub server to communicate to the DNS recursive resolver through QUIC. This is a limitation because QUIC used in this project is a single-threaded library. Linking the two libraries would give the ability to test multiplexing. Multiplexing without head of line blocking means that QUIC servers could accepts queries from different stubs concurrently. This could result in a very interesting timing results for running DNS/QUIC.

7.3 Run QUIC with 0-RTT

QUIC protocol allows running 0-RTT. In the current DNS/QUIC proof of concept test design and implementation,it runs with 1-RTT. Running 0-RTT means transmitting data with no need for the handshake establishment which essentially might be a significant improvement in latency reduction. This could be done by pushing a frame where the FIN bit is set to true. packet to the stream when a DNS response is received, and allocating a new stream for the next incoming query. FIN frame should be pushed from both QUIC endpoints. FIN indicates that the stream that in use should be closed; then, QUIC closes the stream [10]. In the scenario where multiple stubs are communicating to QUIC servers. Then, a unique stream is allocated each stub communicating to the QUIC serves and all future requests will keep using the same stream that was previously allocated to the stub initiating the query, and a random connection ID must be used [12].

7.4 Improve The Encoding of DNS Queries in QUIC

In the current implementation, “www” substring has to be appended to the domain name when initiating DNS query to overcome DNS message corruption. This is not the ideal approach, because sometimes DNS query must not contain the “www” substring. For example running “dig txt _wessam.responsible.ie” in the operating system to get a TXT record of a domain will fail because if “www” substring is appended; then, the DNS server considers the query malformed. A better approach would be search the packet for the different components of DNS header such as questions and record sections. Then, determining the start of the DNS header and remove the unwanted appended data by QUIC servers. Another alternative approach is to precisely determine the length of QUIC packets and find to what extend data could vary in the header from a packet to another. Then, the packet decapsulation and the extraction of the protected payload which is the DNS message will be easier.

7.3 Using a Better Timing Infrastructure

While testing and timing the different protocols, a personal laptop was in use. This may add an overhead and affect timings because of windows managers of the operating system running. A server with a better computation capabilities may be used for testing and timing solely where windows managers ate stopped.

REFERENCES

https://www.scss.tcd.ie/publications/projects/fyp.17/TCD-CS-FYP-2017-1.pdf
https://www.scss.tcd.ie/publications/projects/fyp.13/TCD-CS-FYP-2013-23.pdf
https://www.scss.tcd.ie/publications/projects/fyp.16/TCD-CS-FYP-2016-51.pdf
https://www.scss.tcd.ie/publications/projects/fyp.17/TCD-CS-FYP-2017-12.pdf
https://www.scss.tcd.ie/publications/tech-reports/reports.09/TCD-CS-2009-53.pdf
https://www.scss.tcd.ie/publications/theses/diss/2014/TCD-SCSS-DISSERTATION-2014-028.pdf
http://www.sussex.ac.uk/ei/internal/forstudents/engineeringdesign/studyguides/techreportwriting#1
http://coursepress.lnu.se/subject/thesis-projects/literature-review/
https://datatracker.ietf.org/meeting/97/materials/slides-97-edu-sessc-dns-privacy/

[1] Data Communications and Networking Book
[2] https://tools.ietf.org/html/draft-ietf-dprive-dns-over-tls-09
https://rumyittips.com/what-is-dns-cache-poisoning/
http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch02lev1sec6.html
[3] https://tools.ietf.org/html/draft-ietf-dprive-dns-over-tls-09#ref-BCP195
https://hpbn.co/transport-layer-security-tls/
https://hpbn.co/transport-layer-security-tls/
[4] https://tools.ietf.org/html/draft-ietf-dprive-dnsodtls-15
[4.1] https://libguides.tru.ca/c.php?g=194005&p=4955748
[5] https://datatracker.ietf.org/doc/draft-ietf-quic-transport/?include_text=1
[6] https://tools.ietf.org/html/draft-ietf-quic-tls-09#section-1
[7] https://tools.ietf.org/html/draft-huitema-quic-dnsoquic-00#section-4
[8] http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html
http://www.isg.rhul.ac.uk/~kp/Darmstadt.pdf
https://www.slideshare.net/MenandMice/how-to-send-dns-over-anything-encrypted
http://www.mdpi.com/1424-8220/17/7/1609/htm

https://wiki.aalto.fi/download/attachments/109392027/DTLSPresentation.pdf?version=2&modificationDate=1449662375877&api=v2
https://security.stackexchange.com/questions/29172/what-changed-between-tls-and-dtls
https://en.wikipedia.org/wiki/Denial-of-service_attack
https://tools.ietf.org/html/draft-ietf-dprive-dnsodtls-15
[DNS Header 2 Image]
http://www.inacon.de/ph/data/DNS/Header_fields/Header_section_format/DNS-Header-Field-ANCOUNT_OS_RFC-1035.htm
[10] https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00
[11] https://quicwg.github.io/base-drafts/draft-ietf-quic-transport.html#QUIC-TLS
[12] https://tools.ietf.org/html/draft-ietf-quic-transport-08#section-5.5