ENSC 427: COMMUNICATION NETWORKS

Spring 2023

FINAL PROJECT
ANALYSIS OF CLOUD SECURITY USING TLS/HTTP/TCTP

www.sfu.ca/~alons/ProjectHomepage.html

WRITTEN BY GROUP 3

ALON SINGH	301381523	alons@sfu.ca
RIKU MAKITA	31381399	rmakita@sfu.ca

Table of Contents

1 Abstract

Cloud security solutions commonly use HTTP intermediaries which include reverse proxies, load balancers, and intrusion prevention systems. Which acts as the TLS server connection ends and access HTTP/TLS plaintext to perform their functions. This method chooses its configuration randomly without considering the vulnerability to attacks and outside threats. Further, it has various other shortcomings such as inefficient presentation languages, message flow vulnerabilities and the circumvention of HTTP streaming. Fueled by cloud adoption by large enterprises increasing exponentially, the need for improvement and cunning edge security arises. One of the potential solutions that addresses these issues is the Trusted Cloud Transfer Protocol (TCTP), which applies entity body encryption that can overcome these vulnerabilities. The key concept of TCTP is HTTP application layer encryption channels which integrate TLS functionality into the HTTP application layer. In this project, we will delve deeper into TCTP and other potential methods in relation to cloud security.

2 Introduction

With the evolution of technology and humanity pushing towards complex topics such as Artificial Intelligence, Genomics, Robotics, Quantum Computing and Digital Reality it has become increasingly important for the company developing these new innovations to ensure their data and applications are readily available and secured for authorized users. There must be a reliable method to access, develop, and share this information securely throughout the world. Currently there are a few companies that lead in this industry which include Datadog, Check Point Software, Trend Micro, Crowdstrike, and VMware. We will first discuss what branches incompasses cloud security and what our focus of the paper will be on. Then to our objective to understand how these companies incorporate TLS, HTTP, and TCTP to operate their security software. First we will discuss how TLS and HTTP work and their function within cloud security. Further, we will introduce TCTP which acts on top of the HTTP payload and where its use case can be seen. Finally, simulations will be done to decipher between malicious vs authorized behavior through communication to accurately represent the companies that offer this type of cloud security software to corporations and we will compare simulation time to determine which method is most efficient. Our main source for this project came from an article by M. Slawik, "The Trusted Cloud Transfer Protocol," and also uses information particularly on TLS and SSL from the textbook J. F. Kurose and K. W. Ross, “Computer Networking, 8th Edition” chapter 8.

The paper The Trusted Cloud Transfer Protocol offers great background on the use and purpose of TCTP but goes into very little detail on its performance and integration [1]. Going forward, this paper will take into consideration the suggested further developments made by M. Slawik. Benchmarking the Performance Impact of Transport Layer Security in Cloud Database Systems on the other hand goes into great detail on simulation results, successfully presenting performance benchmarks such as throughput and latency with different encryption techniques.

3 Technical Overview

3.1 Cloud Security Overview

Cloud security is a shared responsibility model where the cloud security provider is responsible for the security of hardware and software while the client is responsible for the security of their own assets. The most important functionality of cloud security is to ensure only authorized users can access the data. Since cloud environments are heavily interconnected, it is very difficult to maintain a secure perimeter. It is always going to be a balance between enabling companies to take advantage of cloud computing benefits while minimizing data security risks. Cloud security includes the following: identity and access management, governance, encryption, disaster recovery, monitoring, and compliance. The paper will delve deeper into encryption which is how cloud security providers scramble data so only authorized parties can send and receive information. Encrypted data is meaningless to hackers unless the decryption key is discovered. Encryption can be done both while data is being stored in the cloud or when it is being transmitted from sender to receiver. There are two main encryption methods in industry both provide a secure communication channel between two applications on the internet. The subsequent section will explain in more detail the (SSL) Secure Socket Layer and (TLS) Transport Layer Security.

3.2 Application Layer Overview

Liked discussed in the previous chapter, the two main encryption methods used in industry today are the Secure Socket Layer and Transport Layer Security. These two methods are both present in the application layer and utilize HTTPS as shown below in figure 1. Starting with SSL, which originated in the 1990s created by Netscape was the first innovative way to secure communications and encrypt messages. SSL begins with a TLS handshake where two communicating parties open a secure connection and exchange a mutual public key. Within this session, a session key is created and this key encrypts and decrypts all communication after the TLS handshake. Each session will have a different key and only the two parties involved have access to the specific key. This makes this communication explicit and requires the client to issue a specific command to the server after the connection. In comparison, the Transport Layer Security uses an implicit connection. The server defines a specific port for the client to be used for the secure connection. Transport Layer Security is known now to be the improved version of Secure Socket Layer and is the industry standard.

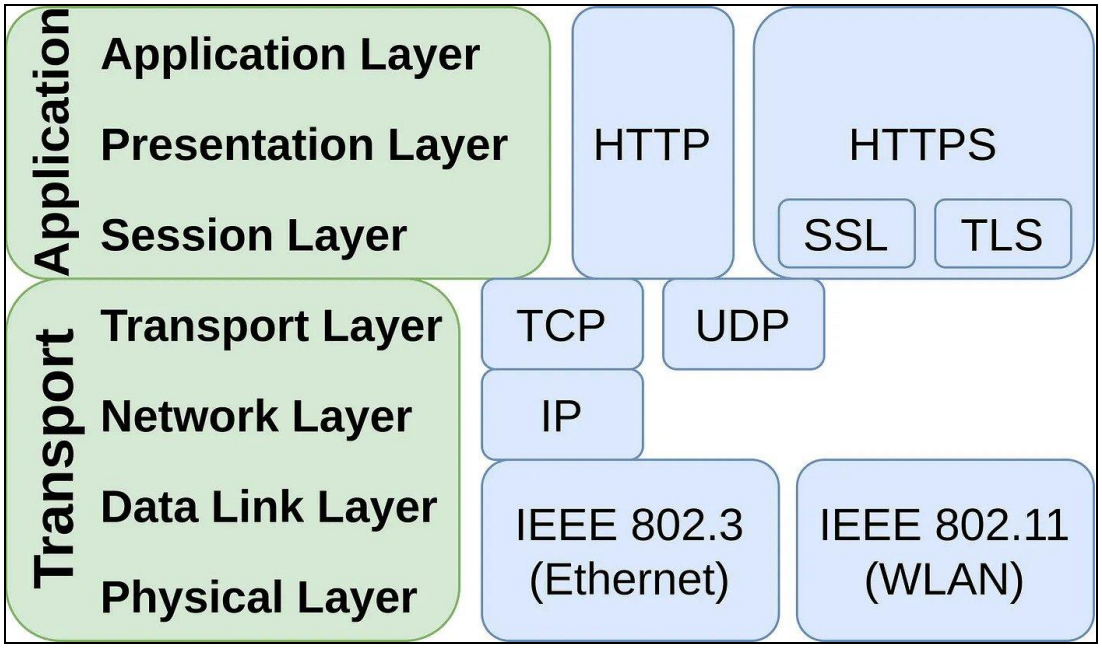

Figure 1: Application Layer

To provide further clarity on these protocol processes, the following is an example of how Transport Layer Security is implemented from the textbook.

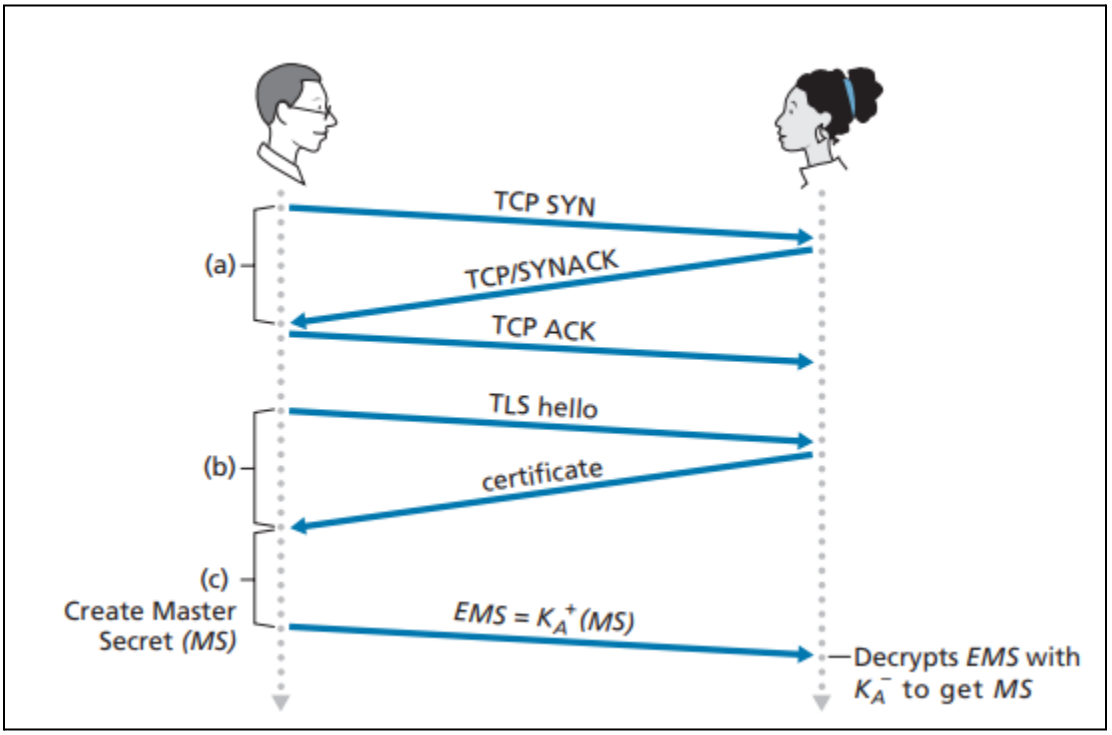

Figure 2: TLS Handshake

In this example, communication is done between Bob and Alice. In stage a, Bob opens up a TCP connection with Alice. In stage b, verification is done to ensure that Alice is Alice and in stage c, Bob will send a secret key to Alice which will be used by both to generate a symmetric key. This interaction is known as the TLS handshake and is the first protocol in TLS communication.

3.3 Trusted Cloud Transfer Protocol

Trusted Cloud Transfer Protocol, also known as TCTP, is an example of a potential innovative new solution that streamlines TLS and SSL. Its method is unique and different and in this section we will discuss the advantages and disadvantages in theory to using this method. How TCTP works is by securing the HTTP payload by encrypting and authenticating it using TLS at the application layer as shown in figure 3. This means that TCTP is cross compatible to hold secure encryption in both the HTTP and HTTPS application layer. This allows cloud computing intermediaries to access all headers while addressing the issues mentioned earlier through entity-body encryption. The negotiation of encryption keys and cipher suites is done by wrapping the TLS Handshake Protocol into the HTTP payload and sending it through intermediaries. By relying on this secure handshake, the risk of intermediaries acting as man-in-the-middle and compromising TCTP security is minimized.

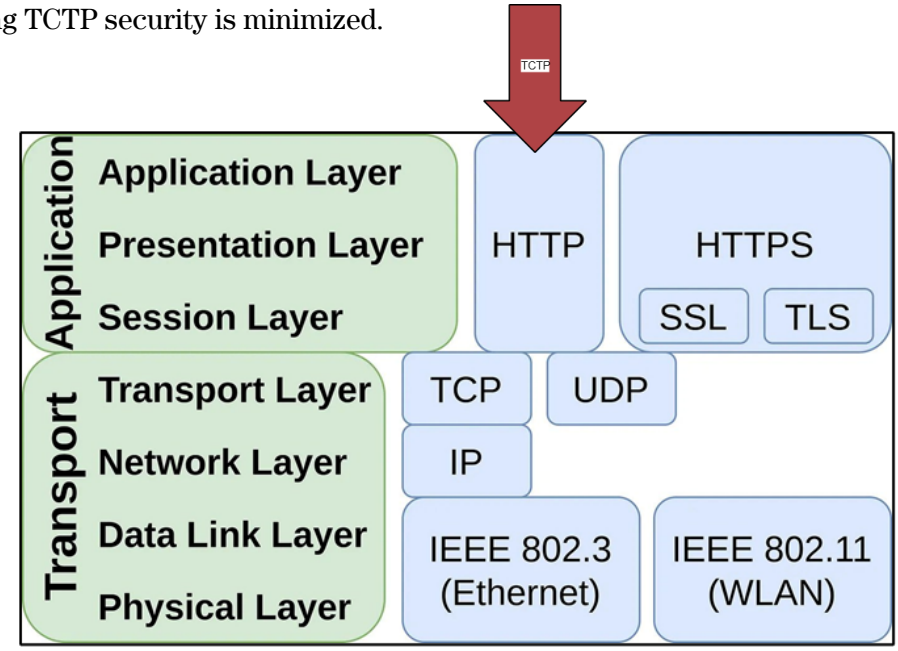

Figure 3: TCTP in the Application Layer

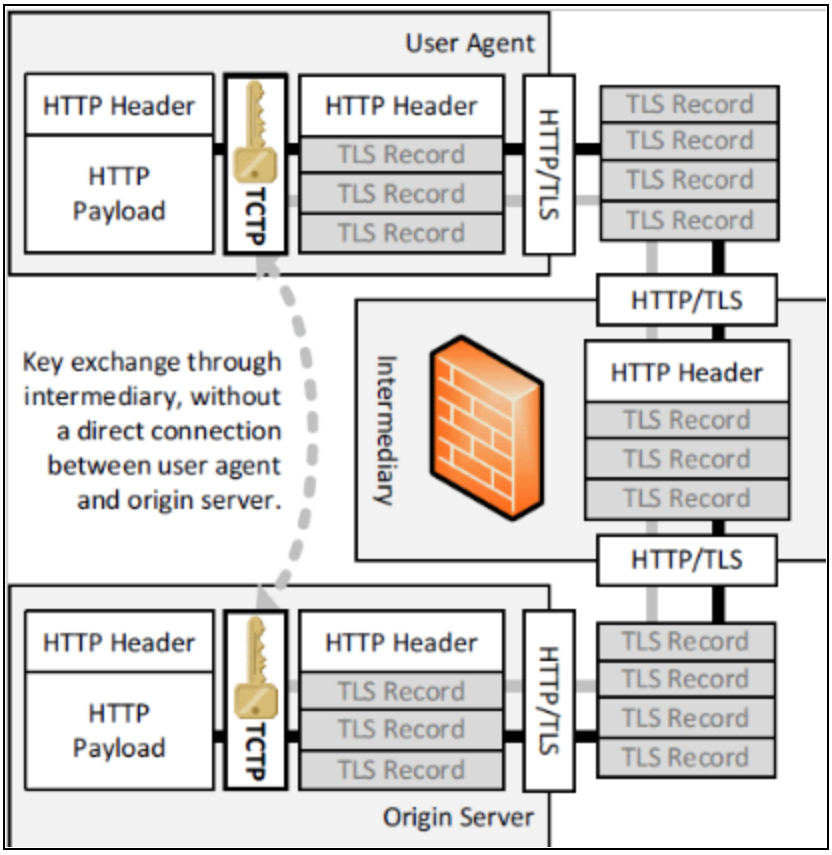

Figure 4: TCTP Flow Chart

"Trusted Cloud Transfer Protocol" (M. Slawik) offers great background on the use and purpose of TCTP but goes into very little detail on its performance and integration [1]. Going forward, this paper will take into consideration the suggested further developments made by M. Slawik. Benchmarking the Performance Impact of Transport Layer Security in Cloud Database Systems on the other hand goes into great detail on simulation results, successfully presenting performance benchmarks such as throughput and latency with different encryption techniques.

The TCTP method of using the TLS header to encapsulate HTTP data would work very similarly to encapsulation where IPv4 addresses are used to surround IPv6 packets making the data receivable by both IPv4 and IPv6 networks. The HTTP header is sent as the payload of an HTTPS/TLS packet containing its own header meaning the data can be read by both HTTP and TLS systems.

4 Simulation

4.1 HTTP - ns3

Although ns3 allows for HTTP simulations, there is not a simple method to incorporate encryption into the simulation the way the HTTPS either TLS or SSL, therefore using ns3 to compare our objects will not produce useful results in a uniform environment. Below is the packet capture of a simple HTTP simulation that will not be compared to any other captures.

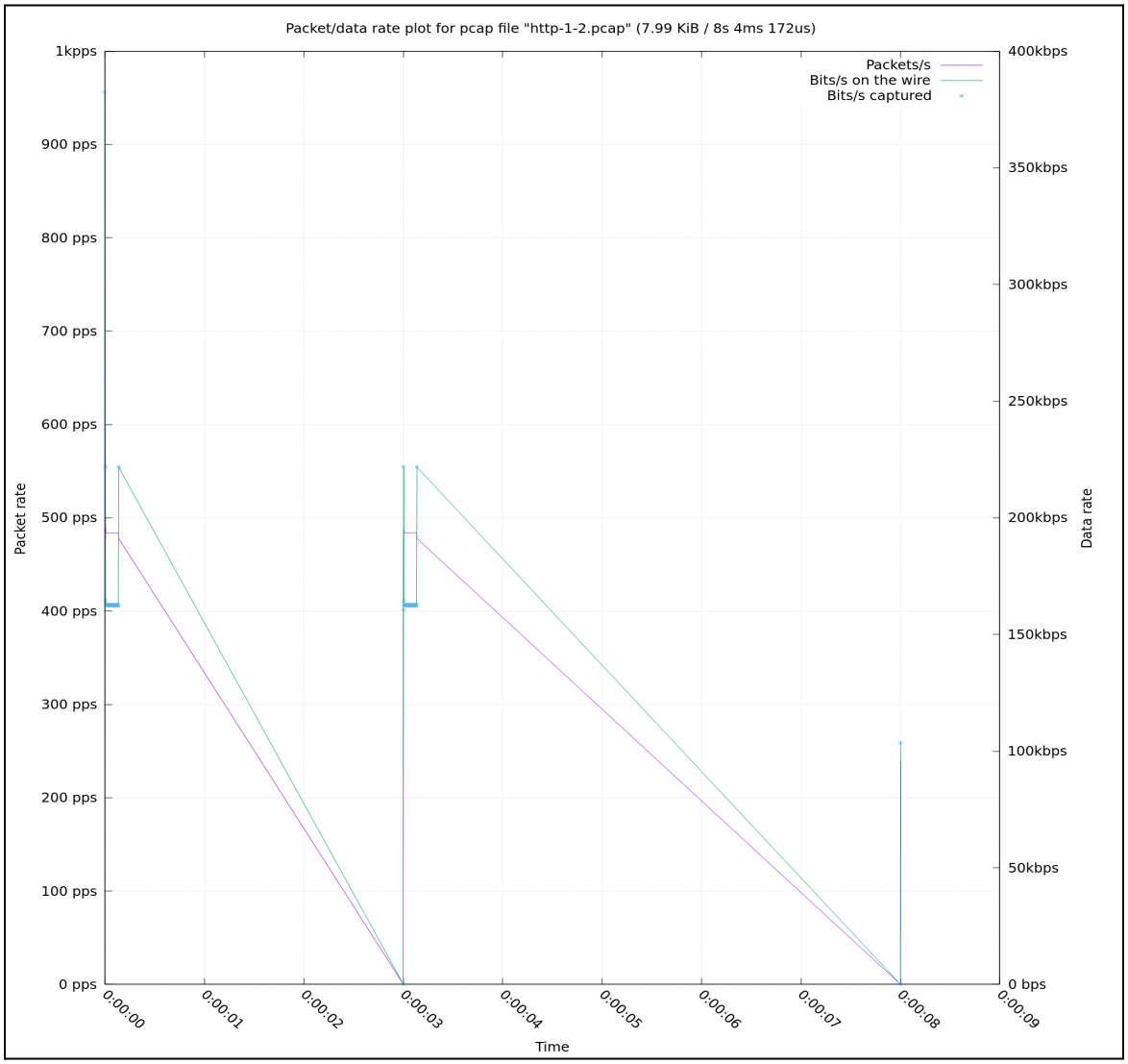

Figure 5: Packet capture of the simulated network

4.2 HTTP - Wireshark

Below are the examples of HTTP data being pulled using the curl method to download site data. This method ensures that the data is not cached in any browser and no additional data is downloaded. The packet captures from the site www.httpvshttps.com [7] . This resource allows us to download 360 non-cached images from both an HTTP and a HTTPS platform.

```
curl http://www.httpvshttps.com
```

Figure 6: Example of the curl command

```
alons@debian:~/workspace/ns-allinone-3.30/ns-3.30$ curl
http://www.httpvshttps.com
<!--
    Created November 2014
    chris@anthum.com
    www.anthum.com
-->
<html>
<head>

    <script src="https://www.httpvshttps.com/check-server.js"></script>

    <script>
       function log(o) {if (console) console.log(o);}
       var proto = window.location.protocol;
       proto = proto.substring(0,proto.length-1);
       function setActiveMenu() {
             if ('http' == proto) {
                   document.getElementById('menu-http').className += '
active';
             } else if ('https' == proto) {
                   document.getElementById('menu-https').className += '
active';
             }
       }
    </script>

    <script>
    ...
```

Figure 7: Example of the return website data

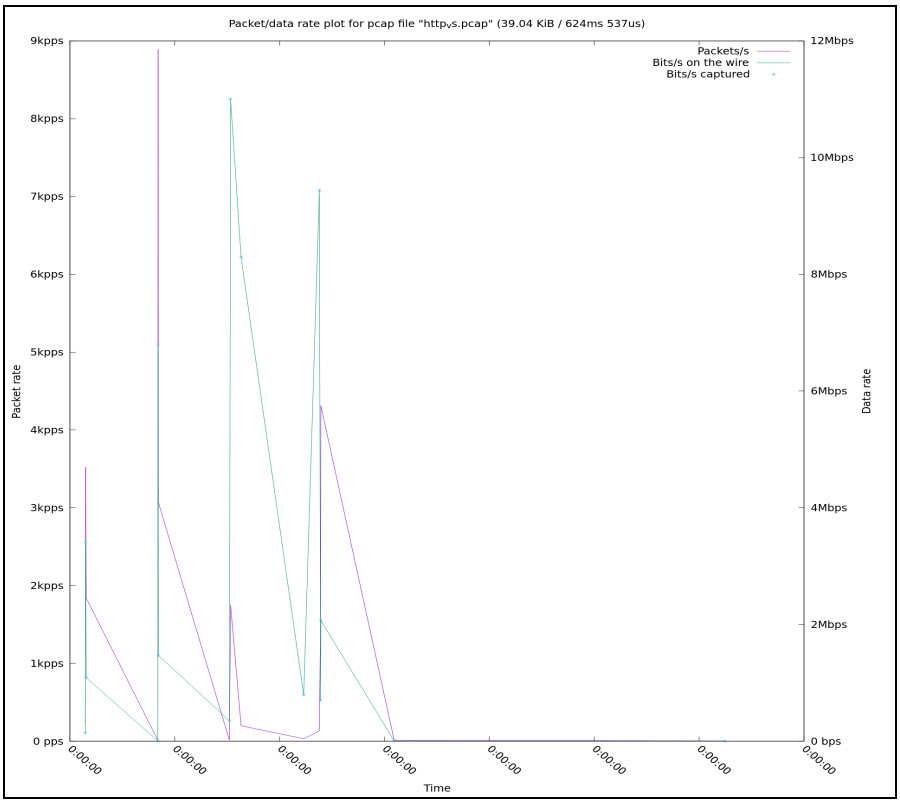

Figure 8: Packet capture of “HTTP vs HTTPS”

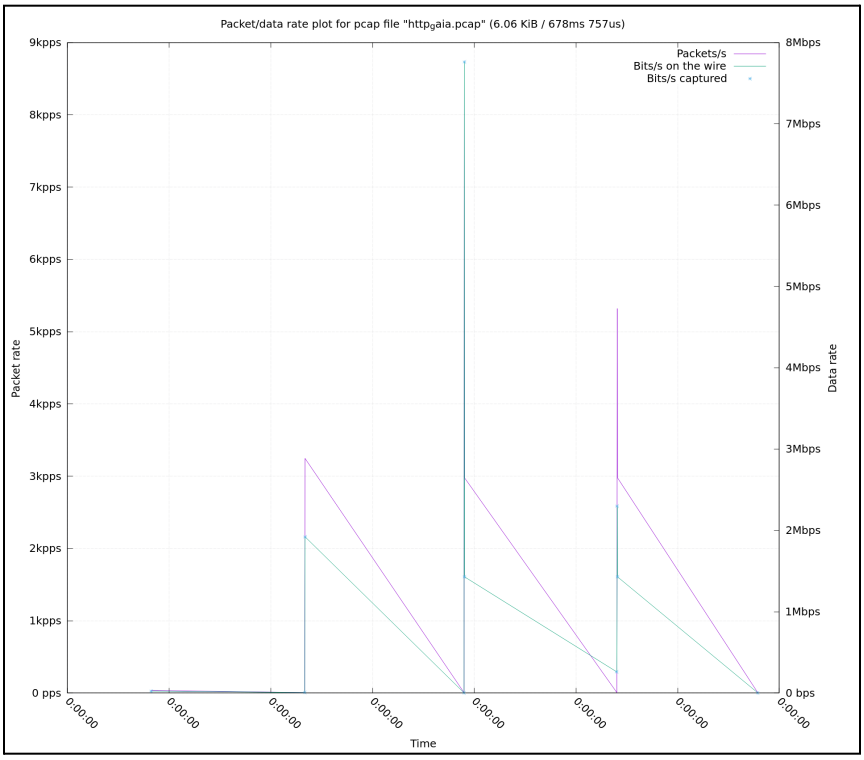

Figure 9: Packet capture of “Bill of Right” [8]

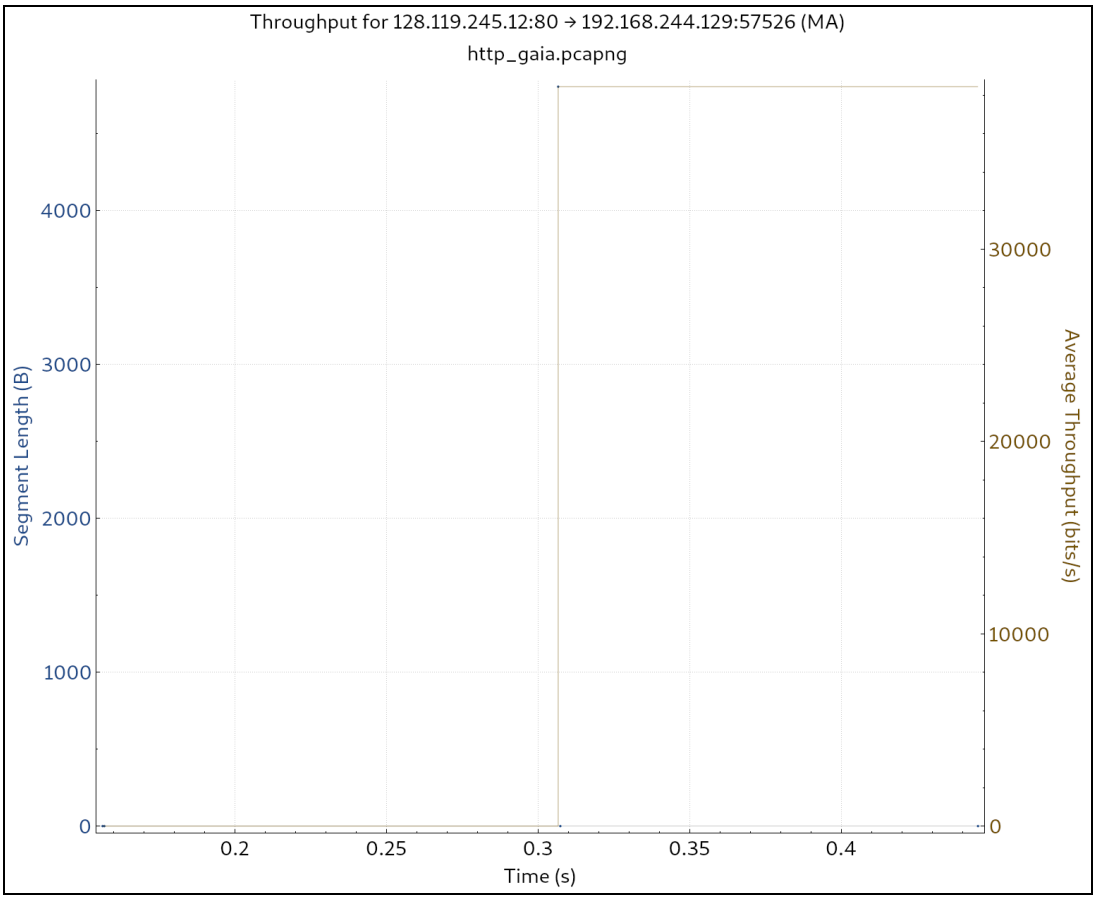

Figure 10: Throughput of the “Bill of Right”

4.3 TLS - Wireshark

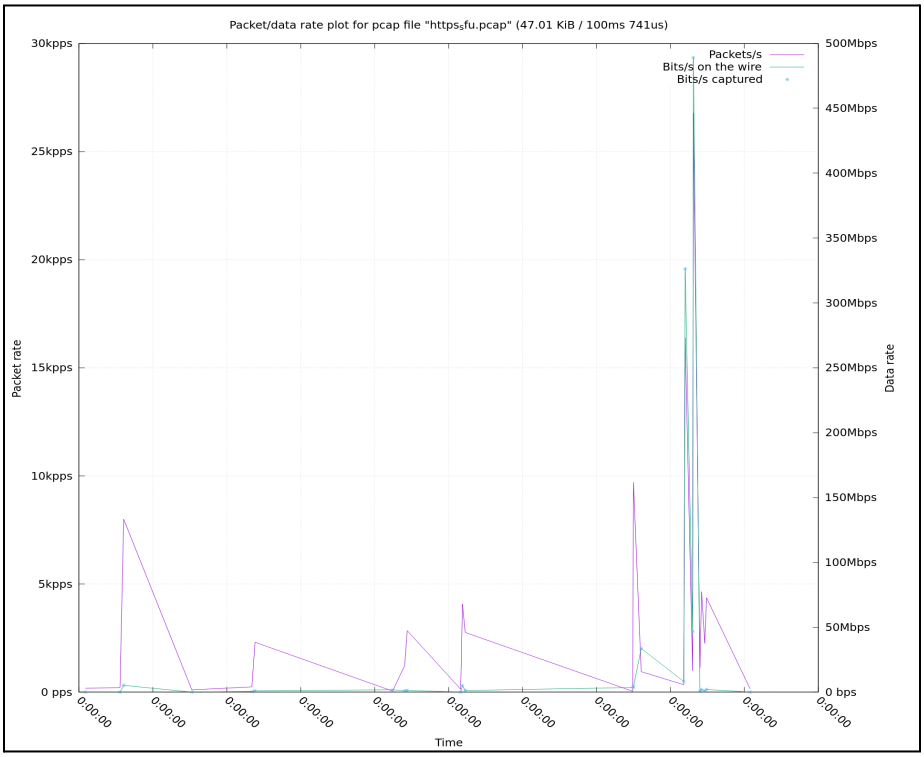

Figure 11: Packet capture of “SFU” [9]

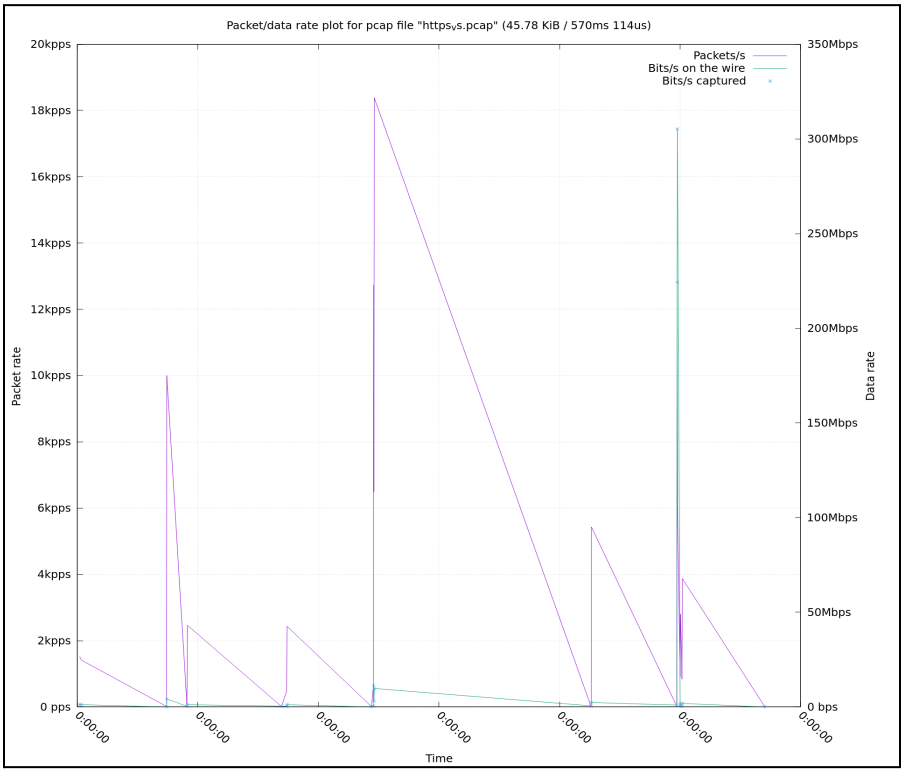

Figure 12: Packet capture of “HTTP vs HTTPS”

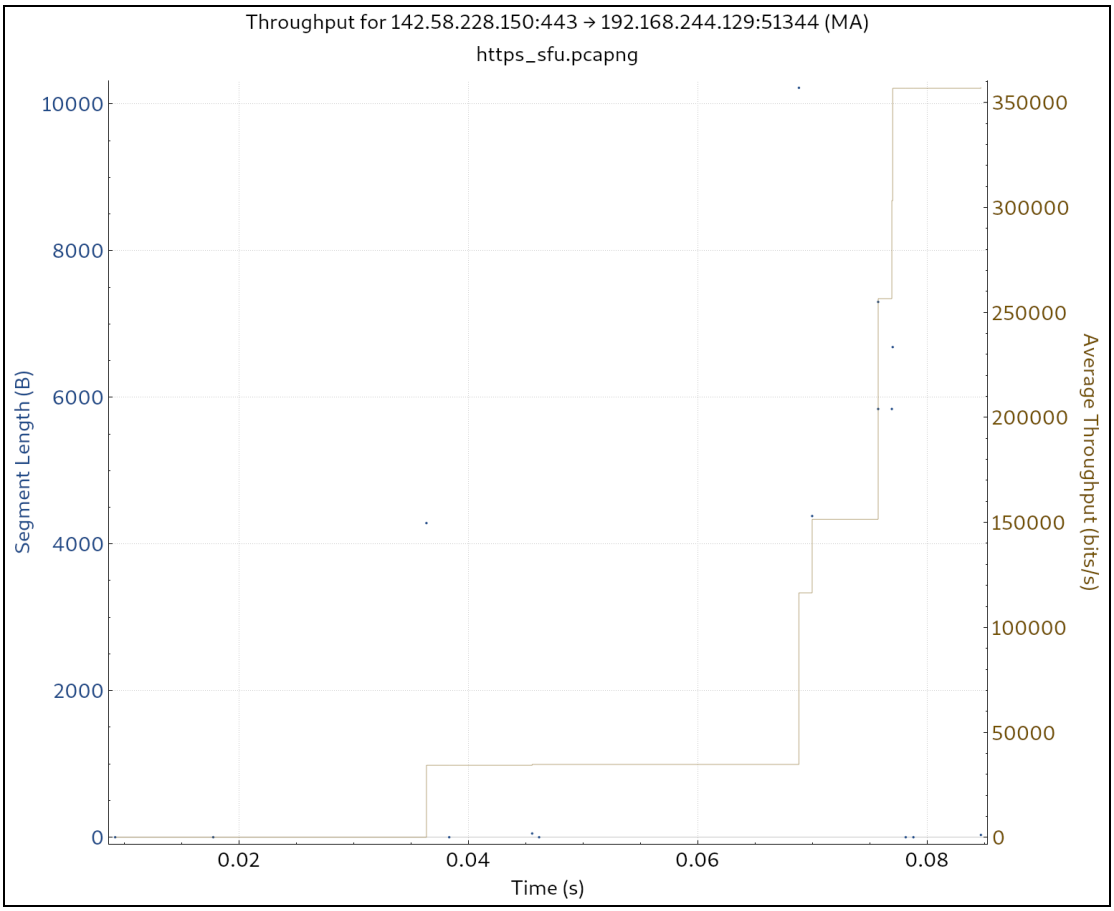

Figure 13: Throughput of “SFU”

5 Discussion

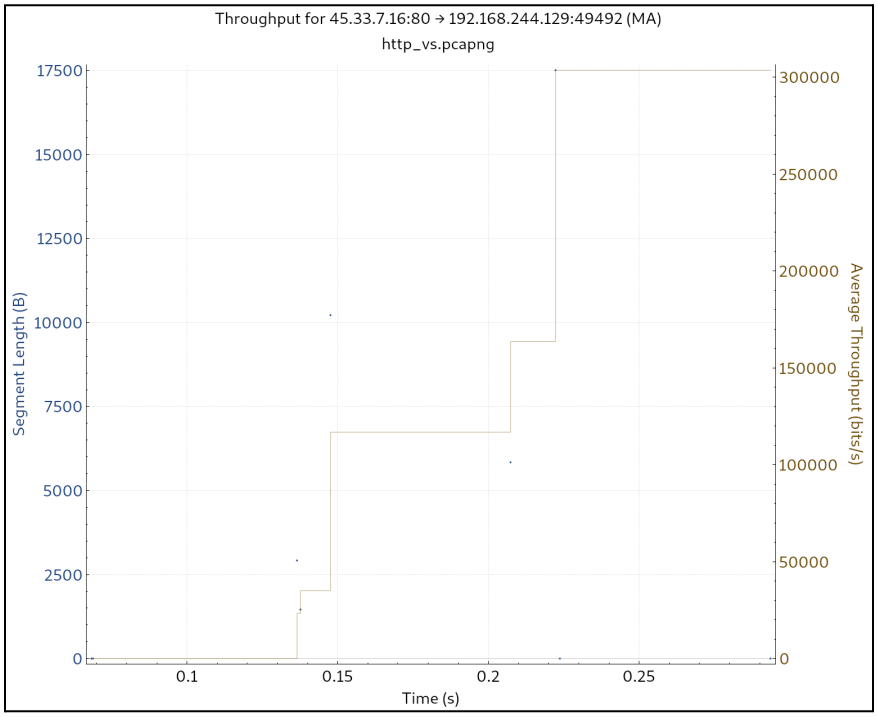

Figure 14: Throughput of HTTP from “HTTP vs HTTPS”

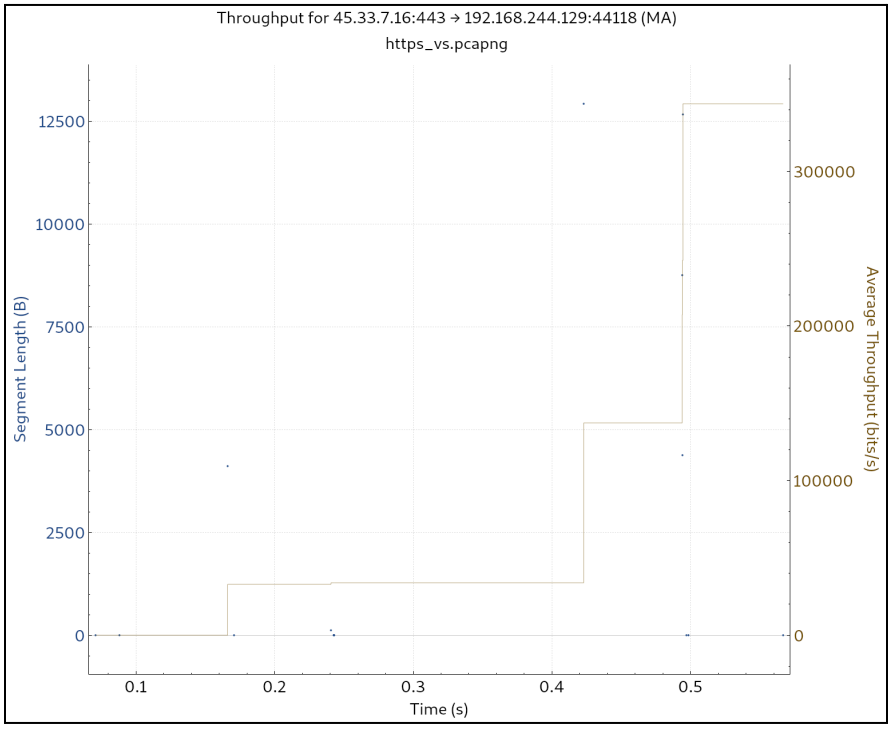

Figure 15: Throughput of HTTPS from”HTTP vs HTTPS”

In figures 14 and 15, we can see that the downloaded HTTP throughout takes ~290ms and that HTTPS takes ~550ms. This means that HTTPS takes almost twice as long to use to download the same data, which makes the most sense since the encryption process will add time to the process. However, programs such as SPDY make HTTPS significantly faster in practice. This can be seen from the “HTTP vs HTTPS” website below.

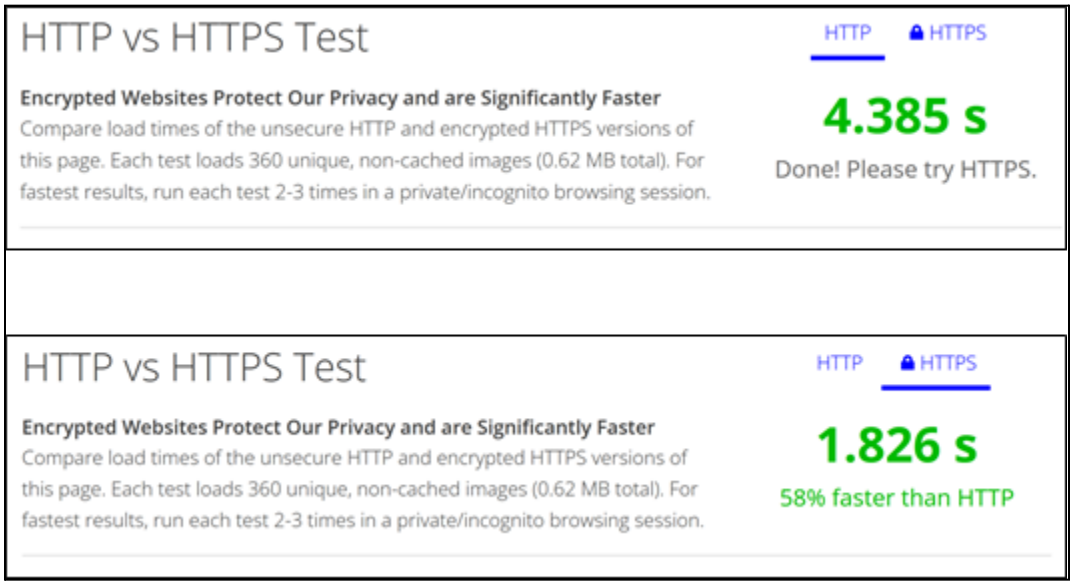

Figure 16: Results from the test site.

SPDY (pronounced "speedy") is a networking protocol developed by Google in 2009 to improve the speed and efficiency of HTTP (Hypertext Transfer Protocol) by reducing the latency and improving the performance of web pages. SPDY is an experimental protocol that has been superseded by HTTP/2 and is no longer actively developed [12].

SPDY operates by multiplexing multiple streams of data over a single TCP connection between a client and a server. This allows for multiple requests and responses to be sent and received simultaneously, eliminating the need for multiple TCP connections and reducing the amount of time required for data transfer.

SPDY also introduces several new features to HTTP, such as server push and header compression, which reduce the amount of data transferred over the network and improve page load times. Server push allows the server to send resources to the client before the client requests them, reducing the number of round trips required to load a page. Header compression reduces the size of HTTP headers, which can account for a significant portion of the data transferred in a web page.

In summary, SPDY is a networking protocol designed to improve the speed and efficiency of HTTP by reducing latency, enabling multiplexing of multiple streams over a single connection, and introducing new features such as server push and header compression. While SPDY is no longer actively developed, its legacy lives on in HTTP/2, which incorporates many of its features and improvements.

6 Conclusion

In conclusion, the comparison of HTTP to TLS and HTTPS highlights the importance of secure communication in today's digital age. HTTP, the widely used protocol for data transfer, lacks security measures that make it vulnerable to cyber attacks. TLS and HTTPS, on the other hand, offer end-to-end encryption, ensuring the privacy and integrity of data being transmitted over the network.

The comparison shows that HTTPS, which combines HTTP and TLS, offers a secure communication channel by encrypting the data being transferred and ensuring that the receiver is authentic. It offers protection against various attacks such as man-in-the-middle, eavesdropping, and data tampering. TLS, on the other hand, offers similar encryption and authentication features but is implemented at a lower level in the network stack, providing an additional layer of security.

Although HTTPS and TLS offer significant security benefits over HTTP, they come at a cost of increased complexity and overhead in terms of processing and computing resources. However, with the advancement of technology and increasing concerns over online security, HTTPS and TLS have become essential tools in securing online communication.

Therefore, the study emphasizes the importance of implementing secure communication protocols such as TLS and HTTPS to protect against cyber attacks. It also highlights the need for organizations to adopt best practices for online security and educate their users to ensure safe online communication.

7 References

[1] M. Slawik, "The Trusted Cloud Transfer Protocol," 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2013, pp. 203-208, doi: 10.1109/CloudCom.2013.126. [Accessed: 26-Feb-2023]

[2] S. Müller, D. Bermbach, S. Tai and F. Pallas, "Benchmarking the Performance Impact of Transport Layer Security in Cloud Database Systems," 2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, 2014, pp. 27-36, doi: 10.1109/IC2E.2014.48. [Accessed: 26-Feb-2023]

[3] M. Msahli, M. T. Hammi and A. Serhrouchni, "Safe box cloud authentication using TLS extension," 2015 International Conference on Cyber Security of Smart Cities, Industrial Control System and Communications (SSIC), Shanghai, China, 2015, pp. 1-6, doi: 10.1109/SSIC.2015.7245679. [Accessed: 26-Feb-2023]

[4] Jabir, Raja & Khanji, Salam & Ahmad, Liza & Alfandi, Omar & Said, Huwida. (2016). Analysis of cloud computing attacks and countermeasures. 1-1. 10.1109/ICACT.2016.7423295. [Accessed: 26-Feb-2023]

[5] Singh, I. D. (2013, December). Data Security in cloud oriented application using SSL/TLS protocol - IJAIEM. Data Security in Cloud Oriented Application Using SSL/TLS Protocol. Retrieved February 27, 2023, from https://ijaiem.org/volume2issue12/IJAIEM-2013-12-10-022.pdf [Accessed: 26-Feb-2023]

[6] Corelight. (n.d.). Corelight/plotcap: Plot packet and data rates over time given a PCAP file, with gnuplot. GitHub. Retrieved April 11, 2023, from https://github.com/corelight/plotcap

[7] HTTP vs HTTPS TEST. HTTP vs HTTPS - Test them both yourself. (n.d.). Retrieved April 11, 2023, from http://www.httpvshttps.com/

[8] National Archives and Records Administration. (n.d.). The bill of rights: A transcription. National Archives and Records Administration. Retrieved April 11, 2023, from https://www.archives.gov/founding-docs/bill-of-rights-transcript

[9] Simon Fraser University. (n.d.). Retrieved April 11, 2023, from https://www.sfu.ca/

[10] Thomas, M. (2021, January 17). HTTPS vs SSL vs TLS. Medium. Retrieved April 11, 2023, from https://medium.com/plain-and-simple/https-vs-ssl-vs-tls-8a0ad0604276

[11] J. F. Kurose and K. W. Ross, Computer networking: A top-down approach. Harlow: Pearson Education Limited, 2022.

[12] “What is Google's SPDY protocol and what does it mean for me and my website?,” *X4B.net*. [Online]. Available: https://www.x4b.net/kb/Legacy/SPDY. [Accessed: 16-Apr-2023].

8 Appendix: HTTP ns3 Simulation Code

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

int main(int argc, char *argv[])
{
    LogComponentEnable("BulkSendApplication", LOG_LEVEL_INFO);
    LogComponentEnable("PacketSink", LOG_LEVEL_INFO);

    // Create nodes
    NodeContainer nodes;
    nodes.Create(3);

    // Create internet stack
    InternetStackHelper internet;
    internet.Install(nodes);

    // Create point-to-point link
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));

    NetDeviceContainer devices1, devices2;
    devices1 = p2p.Install(nodes.Get(0), nodes.Get(1));
    devices2 = p2p.Install(nodes.Get(1), nodes.Get(2));

    // Assign IP addresses
    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.1.1.0", "255.255.255.0");

    Ipv4InterfaceContainer interfaces1 = ipv4.Assign(devices1);
    Ipv4InterfaceContainer interfaces2 = ipv4.Assign(devices2);

    // Create HTTP server and client
    uint16_t port = 8080;

    // Install server app on node 2
```

```
    PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory",
InetSocketAddress(interfaces1.GetAddress(1), port));
    ApplicationContainer serverApps = packetSinkHelper.Install(nodes.Get(2));
    serverApps.Start(Seconds(0.0));
    serverApps.Stop(Seconds(10.0));

    // Install client app on nodes 1 and 0
    AddressValue remoteAddress(InetSocketAddress(interfaces1.GetAddress(1),
port));
    BulkSendHelper bulkSendHelper("ns3::TcpSocketFactory",
InetSocketAddress(interfaces1.GetAddress(1), port));
    bulkSendHelper.SetAttribute("MaxBytes", UintegerValue(1000000));

    ApplicationContainer clientApps1 = bulkSendHelper.Install(nodes.Get(0));
    clientApps1.Start(Seconds(1.0));
    clientApps1.Stop(Seconds(9.0));

    ApplicationContainer clientApps2 = bulkSendHelper.Install(nodes.Get(1));
    clientApps2.Start(Seconds(1.0));
    clientApps2.Stop(Seconds(9.0));

    // Configure tracing
    p2p.EnablePcapAll("http"); // create pcap traces for all point-to-point
links

    // Run simulation
    Simulator::Stop(Seconds(11.0));
    Simulator::Run();

    // Cleanup
    Simulator::Destroy();

    return 0;
}
```