Network Analysis in TLS Protocol Versions

Priyal Shah
shahpri@oregonstate.edu

***Abstract*—Transport Layer Security (TLS) is a cryptographic protocol, built based on its predecessor Secure Socket Layer(SSL). It is the most widely deployed security protocol used today mainly for web services and other client-server applications that require data to be securely exchanged over a network, such as file transfers, VPN connections, instant messaging and voice over IP. The primary goal of this protocol is to provide privacy and data integrity between applications and hence becomes a key part of any data sharing application to handle the various vulnerabilities and security hacks. The objective of this project is to study the different versions of the protocol, exploit their differences, implement the handshake functionality for the 1.x versions and then use them to analyze them based on throughput and communication delay in a simple topology representation.**

I. Introduction

In order to have a secure communication between the clients and the server, a protocol is needed to create a secure environment over the network. TLS protocol is used to provide that secure environment. According to the protocol specification given in [1], TLS is composed of 4 different sub protocols namely the TLS Handshake Protocol,the TLS CipherSec protocol, the Alert protocol and the TLS Record Protocol, as shown in the figure 1. The TLS handshake protocol is

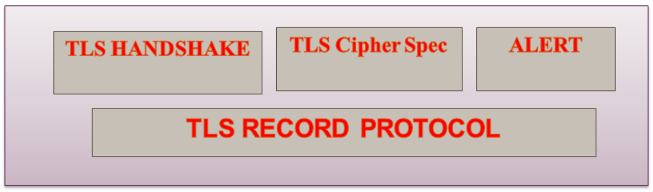


Fig. 1. TLS Handshake Protocol

mainly responsible for containing the details of session ID, key exchange information and also the cryptographic information like hash algorithm. The TLS Cipher Spec is responsible to generate key exchange information and also holds the details of the cipher suite that is common between the client and server for their communication. The TLS Alert protocol is responsible to generate alert messages during the handshake process. For example, when the authentication between the client and the server fails or when the connection is lost during the transmission of data or if the hash algorithm at the client and the server side does not match. The TLS Record Protocol mainly takes the encrypted data from the Application layer and transfers it to the Transport layer. Over the years this protocol has evolved in different versions [2],[3],[4],[5] In this project our focus is to study the TLS Handshake protocol, compare how the handshake process differs in these different versions and analyze how they perform in terms of throughput and delay in a simple topology representation. This report is organized as follows, Section II highlights some of the related works which stood as our motivation to pick this project, Section III briefs the basic definitions as we understood during our study of the TLS Handshake Protocol and the various steps involved during the client server authentication, Section IV discusses our implementation and design decisions for the project along with some key differences in the protocol versions, Section V showcases the setup environment and parameters of interest used for our network evaluation, Section VI runs through our evaluation results and some key observations, Section VII discusses how we are going to use the results observed in our future research and Section VIII concludes this report.

II. Related Work

Comparison of TLS implementations with respect to network protocols, parameters and versions has been a widely explored topic. Analysis of the handshake protocol in terms of security as done in [6] is the base motivation for this project.This paper discusses briefly anout how the handshake protocol performs in terms of the security parameters and its corresponding vulnerabilities. Also papers like [7], [8], [9] analyze TLS in terms of network parameters like throughput, performance which helped us understand the direction we would like to take this work. In addition the presence of different versions of TLS pushed us to compare them while analyzing a few network and security parameters on the TLS Handshake protocol.

III. Study of TLS Protocol and Versions

A. Hash Algorithm MAC/HMAC:

Before the client and server start on the handshake process, they decide on a common hash algorithm. The hash algorithm consists of a hash value that is smaller than the original message and is also unique for every hash algorithm. This unique value can be attached by a code called Message

Authentication Code(MAC). This code helps in checking the integrity of the data that is being transmitted. MAC was being used till the previous versions of TLS 1.0 i.e; SSL 3.0. After the TLS protocol was introduced, HMAC came into existence. HMAC is the Hash Message Authentication Code(HMAC) which does the same function of MAC but also has the shared secret key.

B. Encrypted Pre Master Secret:

Along with the hash algorithm, the client and server decides on a common key encryption algorithm. To encrypt everything with keys and to have a secure connection, it is important to understand the key exchange function called the Handshake to safely agree on a set of keys for both parties to use. Few examples of the key encryption algorithms are RSA, RC4, Diffie- Hellman, Elliptic Curve Diffie-Hellman and the ephemeral versions of the last two algorithms. RSA key encryption is one of the example for asymmetric key encryption in which there are two different keys called the public and private keys which are used for encryption and decryption respectively and vice versa. RC4 key encryption algorithm is an example for symmetric key encryption as it just uses one key to encrypt and decrypt.

C. Initialization Vector:

Different modes of encryption use block of bits to randomize the encryption in order to produce distinct cipher texts. In this process, even if the same plaintext is encrypted multiple times, it can still produce distinct cipher texts. For this entire process, the IV is used. An initialization vector has different security requirements than a key, as IV usually does not need to be a secret. However, in most cases, it is important that an initialization vector is never reused under the same key. The client(encryption) and server(decryption) sides need a copy of the IV to process the handshake function; this IV need not be a secret.

D. Block Cipher and Modes of Operation:

A block cipher is used for secure cryptographic transformation of one fixed-length group of bits called a block. There are different modes of operation. Encrypted Block Chaining (EBC), Cipher Block Chaining (CBC), Plaintext Cipher Block Chaining (PCBC), Cipher Feedback (CFB), Output Feedback (OFB). Cipher Block chaining is an algorithm that uses a block cipher to provide an information service such as confidentiality and authenticity. In CBC mode of operation, each block of plaintext is XORed with the previous cipher text block being encrypted. This way, each cipher text block depends on all plaintext blocks processed up to that point. To make each message unique, an Initialization vector (IV) must be used in the first block.

E. MD5/SHA-1:

There are different hash function algorithms that are used during the encryption and decryption process. They usually differ in their hash values. These has function algorithms are mainly used to check the integrity of the transmitted data. Message-Digest algorithm 5(MD5) is cryptographic hash function -128-bit resulting hash value. In the same way, SHA-1 is Secure Hash Algorithm is also a cryptographic hash function 160-bit resulting hash value. As these help in finding the integrity of the transmitted data, the checksum is calculated at both the ends. The checksum is calculated before transmission of data and after the transmission. If the values turn out to be the same, we can conclude that there was no loss of data during the transmission process.

F. Padding:

A block cipher works on units of a fixed size (known as a block size), but messages come in a variety of lengths. So some modes (namely ECB and CBC) require that the final block be padded before encryption. Several padding schemes exist. Block cipher algorithms like AES and Triple DES in Electronic Code book (ECB) and cipher Block Chaining (CBE) mode require their input to be an exact multiple of the block size. If the plaintext to be encrypted is not exact multiple, you need to pad before encrypting by adding a padding string. When decrypting, the receiving party needs to know how to remove the padding in an unambiguous manner.

G. Pseudo Random Generator:

It is also called as a Deterministic random bit generator which generates sequences of random numbers is generated. For any possible random input, a random output. The pseudo random generator falls in to the family of pseudo random function. All the outputs are random, regardless of how the inputs are chosen.

H. TLS Handshake Protocol

The TLS Handshake protocol is responsible for the authentication and key exchange necessary to establish or resume secure sessions. When establishing a secure session, the Handshake Protocol manages the following:

Cipher Suite Negotiation: The client and server establish a connection and chooses the cipher suite that will be used during the message exchange.

Authentication of Client/Server: The server proves its identity to the client. The client might also need to prove its identity to the server. Public Key Infrastructure(PKI), the use of public/private key pairs, is the basis of the authentication.

Session Key Information Exchange:
The client and server exchange random numbers and also the session key which is used for hashing and encryption.

After building a TCP connection, TLS handshake is started by the client as shown in figure 2 which is derived from the steps given in [10]. The client sends a number of specifications which include the version of TLS running at the client side, which cipher suite is used and also the

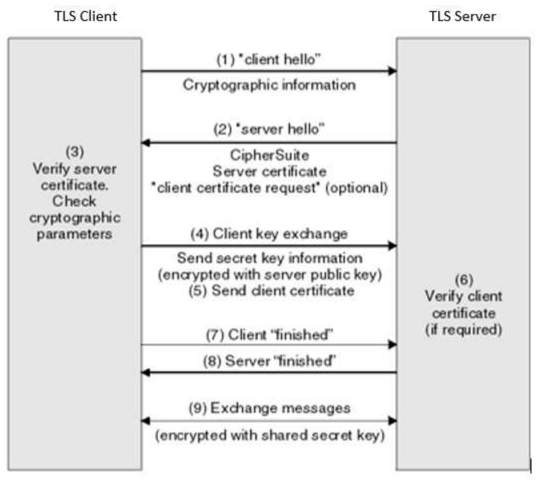

TLS Client
TLS Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

Fig. 2. TLS Handshake Protocol

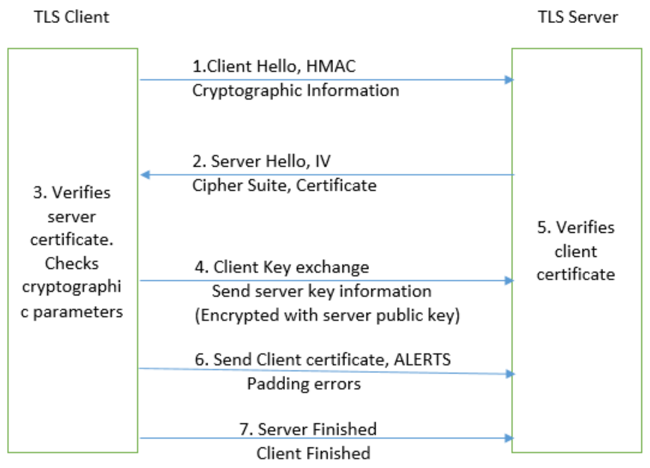

TLS Client
TLS Server
1.Client Hello, HMAC
Cryptographic Information
2. Server Hello, IV
Cipher Suite, Certificate
3. Verifies server certificate. Checks cryptographi c parameters
4. Client Key exchange
Send server key information
(Encrypted with server public key)
5. Verifies client certificate
6. Send Client certificate, ALERTS
Padding errors
7. Server Finished
Client Finished

Fig. 3. TLS 1.0 Handshake Protocol

compression method that will be used along the handshake process.TLS handshake enables the client and server to establish the secret keys with which they communicate. The TLS client sends a ”client hello” message that lists cryptographic information such as the TLS version and, in the client’s order of preference, the Cipher Suites supported by the client. The message also contains a random byte string that is used in subsequent computations. The protocol allows for the ”client hello” to include the data compression methods supported by the client. The TLS server responds with a ”server hello” message that contains the Cipher Suite chosen by the server from the list provided by the client, the session ID, and another random byte string. The server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a ”client certificate request” that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CAs). The TLS client verifies the server’s digital certificate. The TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string itself is encrypted with the server’s public key. If the TLS server sends a ”client certificate request”, the client sends a random byte string encrypted with the client’s private key, together with the client’s digital certificate, or a ”no digital certificate alert”. The TLS server verifies the client’s certificate. The TLS client sends the server a ”finished” message, which is encrypted with the secret key, indicating that the client part of the handshake is complete. The TLS server sends the client a ”finished” message, which is encrypted with the secret key, indicating that the server part of the handshake is complete. For the duration of the TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

I. TLS 1.0

TLS 1.0 was an upgrade from SSL 3.0 during the year 1999. TLS 1.0 uses HMAC instead of MAC as used by SSL for authentication and the CBC mode for key generation. The CBC mode as explained earlier used an implicit Initialization vector to introduce sufficient randomization to the key generation function.Though it is somewhat similar to the SSL 3.0, TLS 1.0 performs better in terms of performance and security due to the additional alert messages support.It also used MD5/SHA-1 for authentication. TLS 1.0 address some of the vulnerabilities of SSL but it still is prone to attacks due to use of CBC mode. Attacks like [11] affect this version of TLS. Figure 3 shows the steps involved in the TLS 1.0 Handshake.

J. TLS 1.1

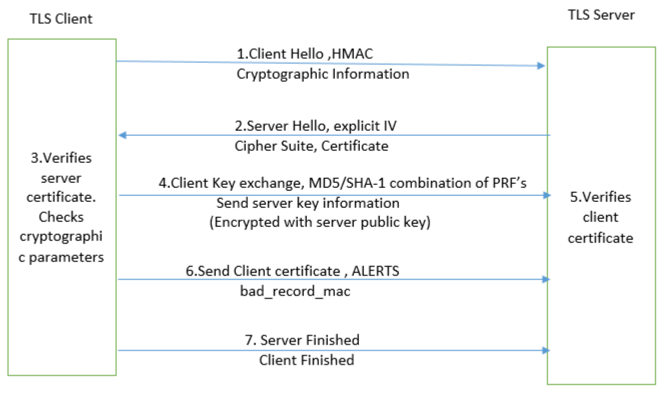

TLS Client
TLS Server
1.Client Hello ,HMAC
Cryptographic Information
2.Server Hello, explicit IV
Cipher Suite, Certificate
3.Verifies server certificate. Checks cryptographi c parameters
4.Client Key exchange, MD5/SHA-1 combination of PRF’s
Send server key information
(Encrypted with server public key)
5.Verifies client certificate
6.Send Client certificate , ALERTS
bad_record_mac
7. Server Finished
Client Finished

Fig. 4. TLS 1.1 Handshake Protocol

TLS 1.1 was introduced in 2006 as an upgrade to TLS 1.0. It was mainly introduced to eliminate the CBC and alert based attacks. Hence in this version CBC mode was not used, instead an explicit IV was used with some additional padding and bad error controls. Though this version was an improvement in terms of security it still was poor in terms of performance due to IV usage.Figure 4 shows the steps involved in the TLS 1.1 Handshake.

K. TLS 1.2

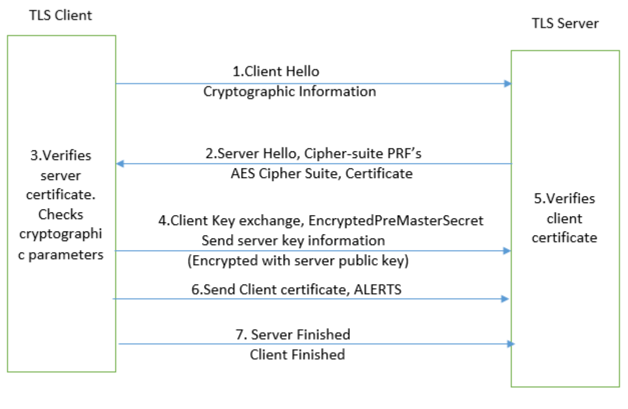

TLS Client
TLS Server
1.Client Hello
Cryptographic Information
3.Verifies server certificate. Checks cryptographi c parameters
2.Server Hello, Cipher-suite PRF's
AES Cipher Suite, Certificate
5.Verifies client certificate
4.Client Key exchange, EncryptedPreMasterSecret
Send server key information
(Encrypted with server public key)
6.Send Client certificate, ALERTS
7. Server Finished
Client Finished

Fig. 5. TLS 1.2 Handshake Protocol

The TLS 1.2 version was introduced in 2008 to eliminate the issues occurring due to key generation. In this version they choose one of 2 modes namely RSA or Diffie Hellman for key generation. By this performance was increased significantly along with security due to the algorithms involved. Here instead of SHA-1 they used SHA-256 since by this time SHA-1 was proved to be vulnerable to various common attacks.This version had significant improvement in terms of both security and speed. Notably TLS 1.2 takes 2 round trips to complete a handshake operation. One limitation of this was that RSA was still prone to various security threats. Figure 5 shows the steps involved in the TLS 1.2 Handshake.

L. TLS 1.3

The TLS 1.3 was a slightly improved version of TLS 1.2 as introduced in 2016. The simple neglect of using RSA and SHA led to this version. Instead Key generation and sharing is done in earlier stages of the communication with use of Ephermal keys as supported by Diffie Hellman and thus increasing both efficiency and security. Moreover compared to TLS 1.2 this version completes in 1 round trip the handshake operation which necessarily halves the latency occurring in TLS 1.2. This happens because the client hello is sent along with the secret key directly and so key generation wait time is completely eliminated from the equation. Needless to mention since key generation has a lot of wait time in TLS 1.0, TLS 1.1 they tend to have high latency values too.

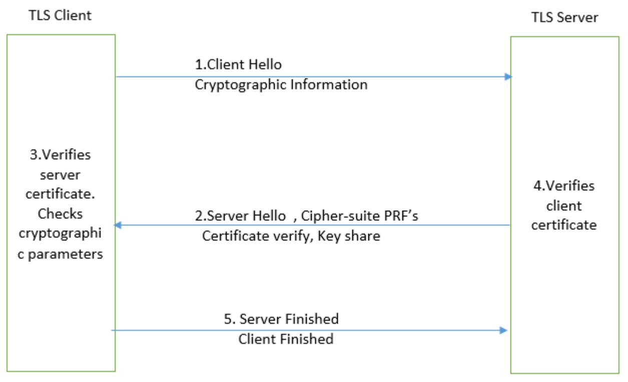
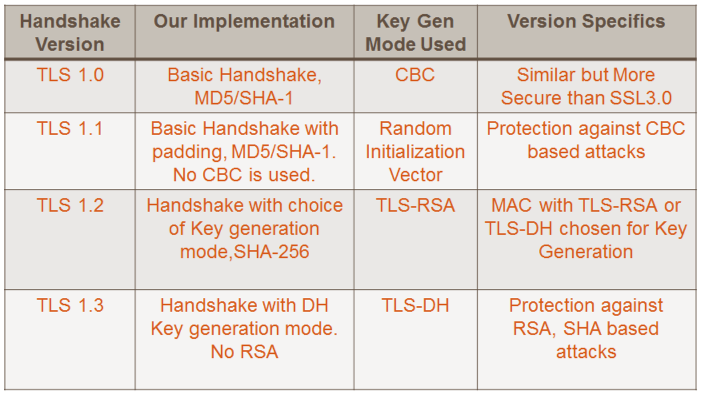

TLS Client
TLS Server
1.Client Hello
Cryptographic Information
3.Verifies server certificate. Checks cryptographi c parameters
4.Verifies client certificate
2.Server Hello , Cipher-suite PRF's
Certificate verify, Key share
5. Server Finished
Client Finished

Fig. 6. TLS 1.3 Handshake Protocol

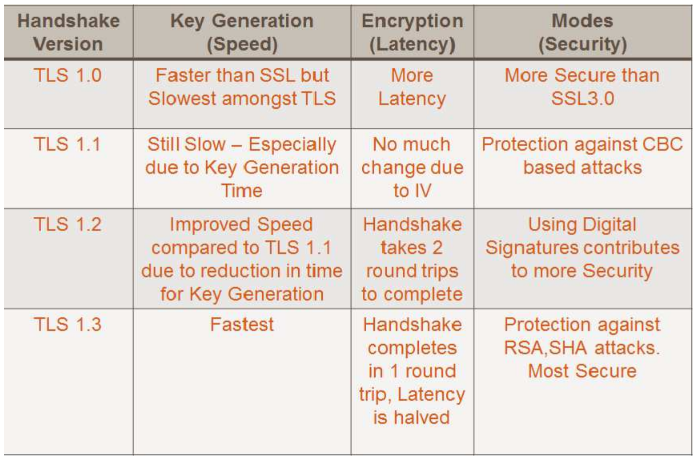
M. TLS Handshake Differences Summary

To Summarize the key differences observed in our study of TLS Handshake Protocol comparing the different versions are as represented in figures 7 and 8



Handshake Version	Our Implementation	Key Gen Mode Used	Version Specifics
TLS 1.0	Basic Handshake, MD5/SHA-1	CBC	Similar but More Secure than SSL3.0
TLS 1.1	Basic Handshake with padding, MD5/SHA-1. No CBC is used.	Random Initialization Vector	Protection against CBC based attacks
TLS 1.2	Handshake with choice of Key generation mode,SHA-256	TLS-RSA	MAC with TLS-RSA or TLS-DH chosen for Key Generation
TLS 1.3	Handshake with DH Key generation mode. No RSA	TLS-DH	Protection against RSA, SHA based attacks

Fig. 7. TLS Versions Conceptual Comparison



Handshake Version	Key Generation (Speed)	Encryption (Latency)	Modes (Security)
TLS 1.0	Faster than SSL but Slowest amongst TLS	More Latency	More Secure than SSL3.0
TLS 1.1	Still Slow – Especially due to Key Generation Time	No much change due to IV	Protection against CBC based attacks
TLS 1.2	Improved Speed compared to TLS 1.1 due to reduction in time for Key Generation	Handshake takes 2 round trips to complete	Using Digital Signatures contributes to more Security
TLS 1.3	Fastest	Handshake completes in 1 round trip, Latency is halved	Protection against RSA,SHA attacks. Most Secure

Fig. 8. TLS Versions Performance Comparison

IV. Implementation

Based on the above described differences we implemented the basic Handshake functionality between clients and server for each of TLS 1.0, TLS 1.1, TLS 1.2 and TLS 1.3 versions using socket programming in Python. We used the cryptographic functions as defined by the libraries described in [12] and [13]. Figure 3, 4, 5, 6 show the steps of handshake functionality that our implementation follows. The results of these implementations were used in our network analysis as seen in the figure 10.

V. Network Evaluation

A. Evaluation Setup

We created an experimental environment in order to measure throughput and communication delay in different versions of TLS by representing it via a simple topology. We used the Mininet Virtual Network Simulator [14] along with the Miniedit GUI support to run our simulation. Firstly, we created the topology with five nodes along with a switch and a controller. A switch can be used as a connection point to every node and controller can be used to pick up random TLS clients in order to get real-time results. Though we represent the switch and the controller to showcase our topology we neglect using them for this scenario. It will be used in future to enhance this project further. However in this scenario, Five nodes are used to represent server and clients. From nodes h1, h2, h3, h4, h5 we implemented h1 as a server with all TLS versions installed on it. After that to get the individual results, we picked, h2 and h3 as TLS 1.0 clients, h3 as TLS 1.1 client, h4, and h5 as TLS 1.2 client and h5 and a TLS 1.3 client. In few cases, client to client communication happens through an intermediate server, in which case we look to demonstrate one of the client acting as a server and the server as the controller that controls their handshake. The corresponding implementations are installed in each of the mentioned nodes. After reading this topology we used iperf to measure throughput and delay between server and client. If there are more than one clients then we measure the throughput and delay between them. The Topology is as represented in figure 9 and 10.

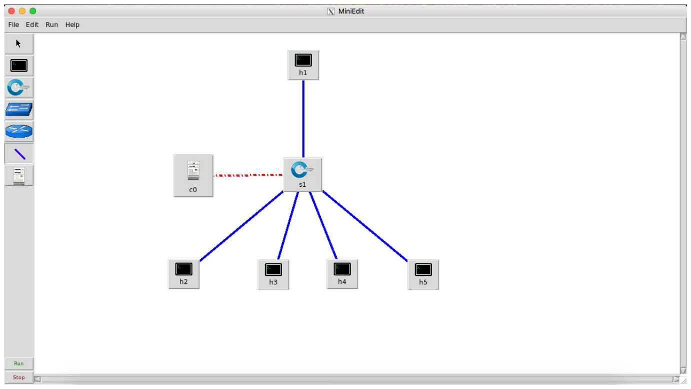
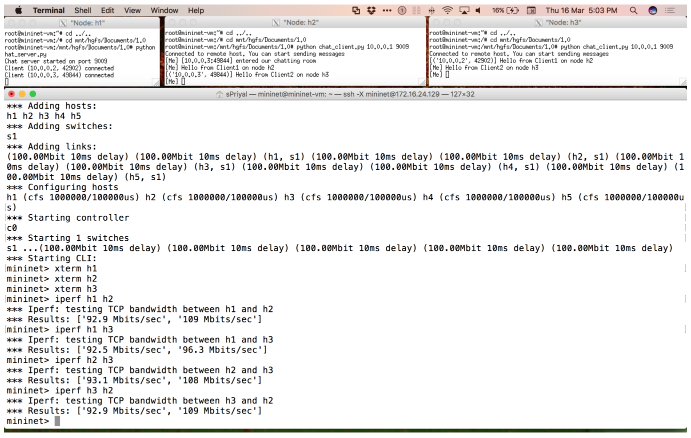


Fig. 9. Network Analysis Topology

B. Results and Discussion

Figure 11 and 12 represents the results we obtained via the simulation as explained above The throughput comparison between the different versions of the TLS Handshake shows that version 1.0 performs better than other versions in terms



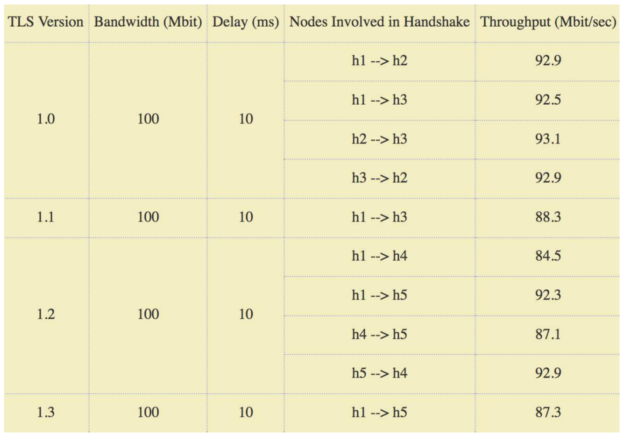
```
"Node: h1"
root@mininet-vm:~# cd ../..
root@mininet-vm:/# cd mnt/hgfs/Documents/1.0
root@mininet-vm:/mnt/hgfs/Documents/1.0# python
hat_server.py
Chat server started on port 9009
Client (10.0.0.2, 42902) connected
Client (10.0.0.3, 49844) connected
```

```
"Node: h2"
root@mininet-vm:~# cd ../..
root@mininet-vm:/# cd mnt/hgfs/Documents/1.0
root@mininet-vm:/mnt/hgfs/Documents/1.0# python chat_client.py 10.0.0.1 9009
Connected to remote host. You can start sending messages
[Me] [10.0.0.3:49844] entered our chatting room
[Me] Hello from Client1 on node h2
[('10.0.0.3', 49844)] Hello from Client2 on node h3
[Me]
```

```
"Node: h3"
root@mininet-vm:~# cd ../..
root@mininet-vm:/# cd mnt/hgfs/Documents/1.0
root@mininet-vm:/mnt/hgfs/Documents/1.0# python chat_client.py 10.0.0.1 9009
Connected to remote host. You can start sending messages
[('10.0.0.2', 42902)] Hello from Client1 on node h2
[Me] Hello from Client2 on node h3
[Me]
```

```
sPriyal — mininet@mininet-vm: ~ — ssh -X mininet@172.16.24.129 — 127×32
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1
*** Adding links:
(100.00Mbit 10ms delay) (100.00Mbit 10ms delay) (h1, s1) (100.00Mbit 10ms delay) (100.00Mbit 10ms delay) (h2, s1) (100.00Mbit 1
0ms delay) (100.00Mbit 10ms delay) (h3, s1) (100.00Mbit 10ms delay) (100.00Mbit 10ms delay) (h4, s1) (100.00Mbit 10ms delay) (1
00.00Mbit 10ms delay) (h5, s1)
*** Configuring hosts
h1 (cfs 1000000/100000us) h2 (cfs 1000000/100000us) h3 (cfs 1000000/100000us) h4 (cfs 1000000/100000us) h5 (cfs 1000000/100000u
s)
*** Starting controller
c0
*** Starting 1 switches
s1 ...(100.00Mbit 10ms delay) (100.00Mbit 10ms delay) (100.00Mbit 10ms delay) (100.00Mbit 10ms delay) (100.00Mbit 10ms delay)
*** Starting CLI:
mininet> xterm h1
mininet> xterm h2
mininet> xterm h3
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['92.9 Mbits/sec', '109 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['92.5 Mbits/sec', '96.3 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['93.1 Mbits/sec', '108 Mbits/sec']
mininet> iperf h3 h2
*** Iperf: testing TCP bandwidth between h3 and h2
*** Results: ['92.9 Mbits/sec', '109 Mbits/sec']
mininet>
```

Fig. 10. Sample Terminal Run



TLS Version	Bandwidth (Mbit)	Delay (ms)	Nodes Involved in Handshake	Throughput (Mbit/sec)
1.0	100	10	h1 --> h2	92.9
			h1 --> h3	92.5
			h2 --> h3	93.1
			h3 --> h2	92.9
1.1	100	10	h1 --> h3	88.3
1.2	100	10	h1 --> h4	84.5
			h1 --> h5	92.3
			h4 --> h5	87.1
			h5 --> h4	92.9
1.3	100	10	h1 --> h5	87.3

Fig. 11. Network Analysis Results

of average throughput in our simulations. These results show why the older versions are still preferred in certain scenarios by servers even though they are vulnerable in terms of security and slow in terms of performance. In terms of security, performance and average throughput all put together TLS 1.2 implementation looks the most effective among the 1.x versions according to our implementation and simulation results.

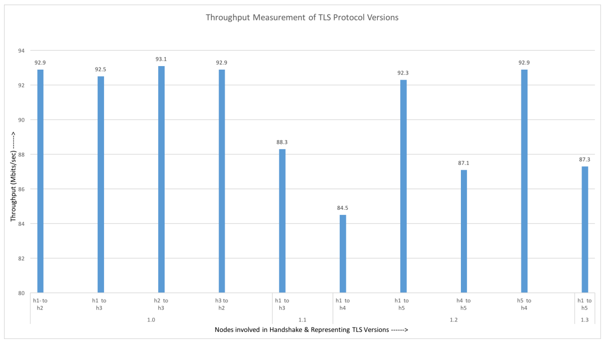


Fig. 12. Graphical Representation of the Results

VI. Future Work

Now that we have a topology set up as mentioned earlier we will try to let the controller choose the server version instead of any assumptive decision on version assignation. It will also be interesting to work around some of the different configuration recommendations given by various authors like [15],[16] and test our implementations using available tools like [17], [18] to try and find more intuitive results on the effectiveness or flaws of our implementation. We would also look to extend this further by including the newer implemented versions of TLS which might change or add to some of our comparisons and results.

VII. Conclusion

TLS, is the standard for secure application network communication both within the enterprise and across the Internet. TLS can help secure your applications by strengthening authentication, encrypting data communications, and ensuring integrity of data in transit. TLS is flexible technologies that adapts well to numerous situations from web server access to server-server communications to virtual private networking. Industry has picked up on this and implemented it widely. It is not always plug and play, but its ubiquitous enough that expertise is fairly common. Finally, there are many versions of TLS protocols that are being launched and new vulnerabilities are introduced as well. But at the end of the day, it is our choice and compromise on to choose the right version of the protocol considering all the factors in order to have a secure communication.

References

[1] C. M. Chernick, C. Edington, III, M. J. Fanto, and R. Rosenthal, "Sp 800-52. guidelines for the selection and use of transport layer security (tls) implementations," Gaithersburg, MD, United States, Tech. Rep., 2005.

[2] T. Dierks and C. Allen, "The tls protocol version 1.0," United States, 1999.

[3] T. Dierks and E. Rescorla, "The tls protocol version 1.1," 2006.

[4] E. Rescorla and T. Dierks, "The tls protocol version 1.2," 2008.

[5] E. Rescorla, "The tls protocol version 1.3," 2016.

[6] P. Morrissey, N. P. Smart, and B. Warinschi, "A modular security analysis of the tls handshake protocol," in *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ser. ASIACRYPT '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 55–73.

[7] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kâafar, "TLS in the wild: an internet-wide analysis of tls-based protocols for electronic communication," *CoRR*, vol. abs/1511.00341, 2015. [Online]. Available: http://arxiv.org/abs/1511.00341

[8] K. Böttinger, D. Schuster, and C. Eckert, "Detecting fingerprinted data in tls traffic," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '15. New York, NY, USA: ACM, 2015, pp. 633–638. [Online]. Available: http://doi.acm.org/10.1145/2714576.2714595

[9] F. Moghimifar and D. Stebila, "Predicting tls performance from key exchange performance: Short paper," in *Proceedings of the Australasian Computer Science Week Multiconference*, ser. ACSW '16. New York, NY, USA: ACM, 2016, pp. 44:1–44:4. [Online]. Available: http://doi.acm.org/10.1145/2843043.2843360

[10] P. Morrissey, N. P. Smart, and B. Warinschi, "The tls handshake protocol: A modular analysis," *J. Cryptol.*, vol. 23, no. 2, pp. 187–223, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1007/s00145-009-9052-3

[11] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the tls protocol," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 62–72. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382206

[12] A. J. Stieber, "Openssl hacks," *Linux J.*, vol. 2006, no. 147, pp. 6–12, Jul. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1145562.1145568

[13] T. P. tlslite at trevp.net, "Tlslite python package," 2015. [Online]. Available: https://pypi.python.org/pypi/tlslite

[14] B. Lantz, "Mininet virtual networks," 2015. [Online]. Available: http://mininet.org/

[15] M. Atighetchi, N. Soule, P. Pal, J. Loyall, A. Sinclair, and R. Grant, "Safe configuration of tls connections," in *2013 IEEE Conference on Communications and Network Security (CNS)*, Oct 2013, pp. 415–422.

[16] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, "Cryptographically verified implementations for tls," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: ACM, 2008, pp. 459–468. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455828

[17] B. Beurdouche, A. Delignat-Lavaud, N. Kobeissi, A. Pironti, and K. Bhargavan, "Flextls: A tool for testing tls implementations," in *Proceedings of the 9th USENIX Conference on Offensive Technologies*, ser. WOOT'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831211.2831212

[18] G. Díaz, F. Cuartero, V. Valero, and F. Pelayo, "Automatic verification of the tls handshake protocol," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04. New York, NY, USA: ACM, 2004, pp. 789–794. [Online]. Available: http://doi.acm.org/10.1145/967900.968063