May 2020

Security Analysis of a Blockchain Network

Naazira Badar Bhat
Clemson University, naazirabhat@gmail.com

SECURITY ANALYSIS OF A BLOCKCHAIN NETWORK

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Naazira Badar Bhat
May 2020

ABSTRACT

Blockchains have gained popularity due to their versatility and wide range of application. Blockchains are a decentralized data structure guaranteeing integrity and non-repudiation of data We use this to secure provenance meta-data. A blockchain can be seen as a distributed database, or a public ledger of transactions or digital events that have occurred and have been shared among participating parties. A consensus is required to verify each transaction. Blockchains are finding use in cryptocurrencies, academics, clinical trials, healthcare and agriculture. However, like other networks, we need to verify the robustness and availability of the blockchain networks. In this thesis, we leverage existing Denial of Service and Distributed Denial of Service[D/DoS] attacks as a tool to evaluate our proposed blockchain technology, *Scrybe*, for robustness. First, we check its performance in presence of Transmission Control Protocol [TCP]-based flooding attacks such as SYN Flooding and its variants. We also optimize TCP kernel parameters to improve the utility of syn cookies as a measure against SYN floods. Second, we evaluate malicious miner attempts to exclude client transactions by stalling the mining process and verify that consensus is reached as long as there is at least one honest miner in the network. The underlying algorithm of *Scrybe* is our novel Lightweight Mining [LWM] algorithm. Our technology guarantees the properties of data integrity and non-repudiation with minimal resource requirements. It introduces a way to mine new blocks in the blockchain, which is not a resource hungry Proof-of-Work [PoW] as required in many present-day cryptocurrency applications.

DEDICATION

Dedicated to my parents *Dr. Badruddin* and *Ms. Waheeda,* and to my grandmother *Ms. Fatima*.

ACKNOWLEDGMENTS

I want to begin by saying that I'm grateful to God for blessing me constantly and helping me reach my goals.

I wish to express my sincere appreciation for my supervisor, *Dr. Richard Brooks*, who convincingly guided and encouraged me to complete this project. Without his help, this thesis wouldn't have reached its ultimate conclusion. Working with him has been a valuable experience. I also wish to show gratitude to my other committee members *Dr. Richard Groff* and *Dr. Adam Hoover* for their constant help and support towards completion of this work. Having them on my thesis committee was a wonderful learning experience.

I would like to express my gratitude to our research team working with Dr. Brooks at Clemson University, and to Dr. Anthony Skjellum and his research group at University of Tennessee Chattanooga, for their invaluable help and support. I am also appreciative of Clemson University's research facilities and other staff who helped me every step of the way to reach this goal.

I also want to profoundly appreciate my parents *Badruddin* and *Waheeda*, my siblings *Bazigha* and *Fawzaan*, my nieces *Ayesha, Fatima* and *Hayah*, my parents-in-law *Gh. Nabi Aali* and *Hafizah,* my siblings-in-law *Sami, Arooge* and *Hafsa,* and my friends for their invaluable love, support, care, and prayers to help me through all my endeavors.

Finally, I want to express my deep gratefulness for my husband *Itmenon,* who not only loves and supports me, but also helps me become a better version of myself every day. It wouldn't have been possible to complete this thesis without his help.

TABLE OF CONTENTS

Table of Contents (Continued) Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Blockchain technology has drawn attention from a wide range of stakeholders including academics, healthcare, and government agencies. A blockchain is a data structure for a shared, distributed, and fault-tolerant database that every participant in the network can access, but none can tamper with. Blockchains are designed to tolerate possibly malicious nodes being present in the participating network, but they rely on honest nodes to ensure that information is not vulnerable to manipulation. The absence of centralization speeds up the entire process. Owing to the cryptographic structure of the blockchain, it is impractical to alter it. Based on these features, blockchains have acquired an enormous range of applications even in sensitive fields. An important aspect for which blockchain-based networks need to be tested is their robustness and availability [1]. D/DoS attacks can be used as a tool to verify these attributes. D/DoS attacks are launched by cyber-criminals using one or more computers against a single or multiple computers or networks. These attacks disable computers, deny access to authorized users of a particular website or an application; or use a breached computer as a launch point for other attacks. A denial of service attack can be an attempt to flood a network, thereby, reducing a legitimate user's bandwidth, preventing access to a service, or disrupting service to a specific system or a user. As a smarter strategy, the attackers target the nodes that utilize the maximum share of the resources. If successful, the data on the application can be useless and unavailable to its owners.

Our research team has proposed a Blockchain technology, *Scrybe*[2] with a novel, light-weight mining algorithm [LWM], for secure provenance and related applications such as clinical trials, academic integrity and digital forensics. The blockchain algorithm employed by most cryptocurrencies like Ethereum and Bitcoin, and other digital services for expanding their blockchain is resource intensive. *Scrybe* offers an alternative which requires fewer resources than proof-of-work, and is more economical. Our proposed technology maintains non-repudiation and data integrity.

The system is built on Transmission Control Protocol (TCP) Client-Server. TCP is a core protocol of the Internet protocol suite. It sits on IP layer, and its primary task is to provide a reliable and ordered communication channel between applications on networked nodes. TCP works in the Transport layer, which provides host-to-host communication services for applications.

In this research, we evaluate *Scrybe's* performance in presence of the TCP protocol-based D/DoS attacks to verify robustness and availability of our algorithm. A SYN Flood D/DoS attack on *Scrybe* is implemented. The attack uses IP-Spoofing, Randomized IP Spoofing and the local area network denial-of-service(LAND) techniques on the blockchain network; and the performance is evaluated in the presence of those attacks. Impact of increasing the size of the attack packets is also studied. We also evaluate the effect of TCP Linux kernel parameters such as the syn-ack retries and the max-syn-backlog queue on the impact of the attack on the network. The tool used to generate the attack traffic is hping3, and the network analyzer used is Wireshark. In addition, we consider the possibility of malicious intent of one or more miners to stall the LWM to prevent the addition of a new verified block to the blockchain, or to deny transactions from some clients to the blockchain. Modifications to the algorithm are suggested to mitigate the impact of anomalous behavior in our system.

Our work focuses on the following:

1. Evaluating the performance of the proposed system during presence of a TCP vulnerabilities like SYN-Flooding attack and its variants, thereby verifying robustness and availability.

2. Verify that consensus is reached even if malicious miners try to stall the mining process as long as at least one miner is honest.

The rest of the document is as follows: In Chapter 2, we provide background information on necessary topics relevant to this work and the related work; in Chapter 3, we describe our experimental set-up and the work done including the results , Chapter 4 has the analysis, and optimization of kernel parameters; and in Chapter 5, we provide a conclusion of the work and some prospects from future work.

CHAPTER 2

Background

Since Satoshi Nakamoto published his Bitcoin white paper in 2008, blockchain has become one of the most frequently considered solutions for ensuring security of stored data and its transfer through decentralized, peer-to-peer networks. Being a cryptographic-based distributed ledger, trusted transactions are enabled among untrusted participants in the network using the blockchain technology. Since the first Bitcoin blockchain was introduced in 2008[3], blockchain systems such as Ethereum and Hyperledger Fabric, have been developed. Blockchain has seen a surge in interest among researchers [[4],[5],[6],[7]], software developers and industry practitioners due to the immutability it offers [[8],[9],[10]].

An important aspect of blockchain technologies is consensus. A majority of the participating miners need to approve who produces the next block and verify/validate this block when added to the blockchain. Bitcoin uses the proof-of-work (PoW) mechanism where miners compete to solve a mathematical puzzle. The miner who solves the mathematical problem earliest gets rewarded with bitcoins. This motivates participants to be honest. However, PoW consumes a lot of energy and is slow. An alternative consensus mechanism which is widely used is proof-of-stake (PoS). In this mechanism, miners who have a financial stake in the network are allowed to produce the next block. Instead of spending energy to validate the block, participants need to prove that they have network tokens in their wallet. Ethereum used PoW, but is now planning a shift to PoS as CasperPoS, despite being energy efficient and less costly than PoW can create an oligarchy. Miners might buy a large number of coins at the beginning of the process and monopolize future mining rounds since they have a larger stake in the network. Variants of PoS such as Proof of

Elapsed Time(PoET), Delegated PoS, proof-of-work, proof-of-importance , proof-of-capacity also exist [11].

Scrybe, on the other hand, is more energy efficient and economical than PoW, thus an attractive choice to reach consensus about who would create the next block. *Scrybe* is described in Section 2.1.

2.1 Scrybe : Our Proposed Blockchain Technology

The Scrybe architecture is provided in Figure 2.1.

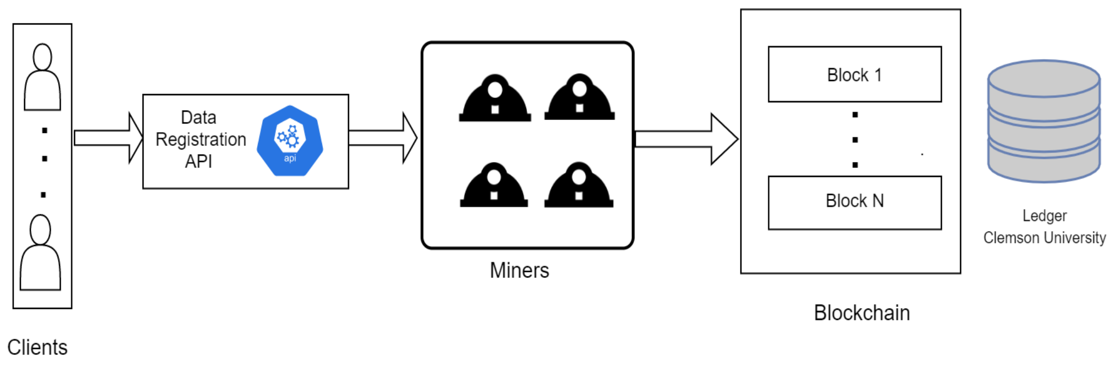


Figure 2.1. The Scrybe architecture

The description of the architecture components is as follows:

A. Blocks

Blocks are one of the prime components of the system. A sequence of verified blocks forms the blockchain. The current block includes hash of the previous block. Hash is a mathematical one-way function - practically infeasible to invert- that maps an arbitrary -size input data to a bit string of a constant size. This property makes the blockchain immutable. Blocks are added to the blockchain by miners or entities authorized to participate in the Light Weight Mining [LWM] round. Miners receive the list of transactions from clients or users, aggregate those and calculate the Merkle root. A Merkle root is the hash of all the hashes of all the transactions submitted to the blockchain network. This allows miners to confirm inclusion of all the transactions in the block by the miner who gets to produce the next block. Once the miner gets selected to add the next block to the blockchain based on the LWM algorithm, they broadcast the block to all other participating miners of that particular round, and the contents of the block including the previous hash, the Merkle root and the miner's signature are verified. This ensures that all transactions are included and that an authorized miner produces the block with a valid signature.

B. Transactions

Transactions are data elements stored on the blockchain. These are the backbone of provenance. Transactions can reference previous transactions if they are not the first transaction themselves, or they can be genesis events, i.e. the first data collected from a particular use case of our blockchain network. Input fields are used to make references to the previous transactions, while as for the genesis event, output fields are used. Normal transactions have both an input and an output field, but genesis events only have an output field. Input fields refer to the hashes of pointers of previous transactions, while output fields contain persistent URLs (PURLs) pointing to the data, along with the SHA-3 hash of the data, ensuring its validity. Additionally, output fields contain PURLs that point to the provenance of the data along with its SHA-3 hash. The size of the blockchain can drastically be reduced by storing the SHA-3 hash of the transaction instead of the transaction itself. The transaction itself will be stored on a transaction server, which is locally maintained, along with the data server and the metadata server which store the relevant information.

C. Light Weight Mining

Our novel Lightweight Mining (LWM) algorithm guarantees data integrity and non-repudiation with minimal resource requirements. It introduces a way to mine new blocks in the blockchain, which is not a difficult Proof-of-Work (PoW) required in present-day cryptocurrency applications[3]. Our approach to blockchain-based data provenance, paired with the LWM algorithm, provides the framework for key classes of provenance to be managed. The algorithm requires minimal processing power and cost. The algorithm is:

Algo 1: Lightweight Mining (LWM)

```
Lightweight Mining Algorithm (LWM)
Input: The number of miners $N$.
Algorithm: For each miner $m_i$, $1 \leq i \leq N$,
  • Step 1: $m_i$ generates a random number $s_i$ ;
  • Step 2: $m_i$ broadcasts the SHA-3 hash of the $r_i$, denoted by $H(s_j)$;
  • Step 3: Once $m_i$ has collected all $N$ hashes $\{H(s_1), H(s_2), \cdots, H(s_N)\}$, $m_i$ broadcasts the random number $s_i$.
  • Step 4: Once $m_i$ has collected all $N$ random numbers $\{s_1, s_2, \cdots, s_N\}$, $m_i$ calculates $l = \sum_{j=1}^{N} s_j \bmod N$.
  • Step 5: $m_l$ is the selected miner to create the next block from the collected transactions. (Without loss of generality, we map $m_i = j$, $1 \leq j \leq N$ as a simple rank ordering for the registered miners.)
```

LWM ensures randomization in selecting the miner for producing the next block. One of the core ideas of LWM is sharing the hash first and then the actual secret random number. This guards against the possibility of a malicious miner exploiting the miner-selection process in their favor, as discussed in Chapter 5. A malicious miner can wait to receive everyone else's random number to decide whether or not they like the outcome of the LWM if they shared their own number. Sharing the hash first ensures that every miner shares the hash of their random number with all other miners before they are able to see other miners' choices. Since a different random number requires a different hash than the one they previously shared, the malicious miner cannot change the random number after sharing its hash without being detected. Thus, LWM can tolerate collusion of up to N - 1 malicious miners. As long as there is one honest miner value among the values used, the modulo operation is randomized.

D. Servers

The servers are locally maintained; and will hold the raw data comprising the ledgers in which the blockchains are held. The integrity of the transaction server can be verified by comparing the list of transactions on the blockchain to that on the transaction server. There should be no discrepancy between the two. Similarly, there should be no discrepancy between SHA-3 hash of the data and the SHA-3 hash stored in the transaction to verify that the data and metadata are disreputable.

2.2 Configuring Scrybe

For implementing *Scrybe,* we pull the git repository of our project on the nodes. After installing all the necessary files, we configure some nodes as miners which receive transactions from the nodes which send those, and some nodes as clients who have files which they need to submit to the blockchain to secure those. The steps are as follows:

a. A node, which acts as a miner, runs the *Scrybe* executable to initiate mining. We have opened up a specific port -10987 where the *Scrybe* miners listen and mine.

b. Another node which wants to join the mining network uses the public IP address of the first miner to join the network. Same is true for the subsequent miners -They join the network using the public IP address of any of the existing miners, and start mining.

c. On each of the miner nodes, the LWM runs and a miner gets selected to produce the next block.

d. A node which is supposed to act as a client first needs to join using the public IP address of any miner.

e. The client first needs to request a blockchain update which provides it the peer list of the participating miners.

f. The client submits transactions to all the participating miners of the LWM round simultaneously.

g. The result of the modulo operation determines who gets to create the new block, adding the client transactions submitted since the last block was added. The block is then shared with all the participating miners, who verify it and add to their own copies of blockchain.

If a new miner needs to join the network, the steps *b-g* are applicable. However, to the new miner, the relevant blockchain update as mentioned in step *e* is the updated length of the blockchain mined so far. This is important before the miner is able to participate in the subsequent LWM rounds because we want the miner to be able to add the next block at the current length of the blockchain produced so far.

2.3 Communication Protocol In Scrybe

In *Scrybe,* the communication protocol used is Transmission Control Protocol (TCP). We have a specific port – 10987 for our application. TCP works with the Internet Protocol (IP). IP defines how packets of data are exchanged between computers. The basic rules are defined by the TCP/IP. TCP is defined in the Request for Comment (RFC) number 793 by the Internet Engineering Task Force (IETF)[16].

Since we exploit TCP vulnerabilities, we elaborate on this protocol. TCP is connection-oriented, i.e. in order to be able to exchange data, a connection needs to be established. Once established, the connection needs to be maintained until the application programs have finished exchanging messages. It determines splitting application data into packets which the networks can deliver. This protocol sends packets to, and accepts packets from, the network layer and manages flow control. It also handles the retransmission of dropped packets as well as manages the acknowledgement of packets arrival at each end. This helps provide an error-free data transmission [12]. TCP format is shown in Figure 2.2. Its size is 32 bits.

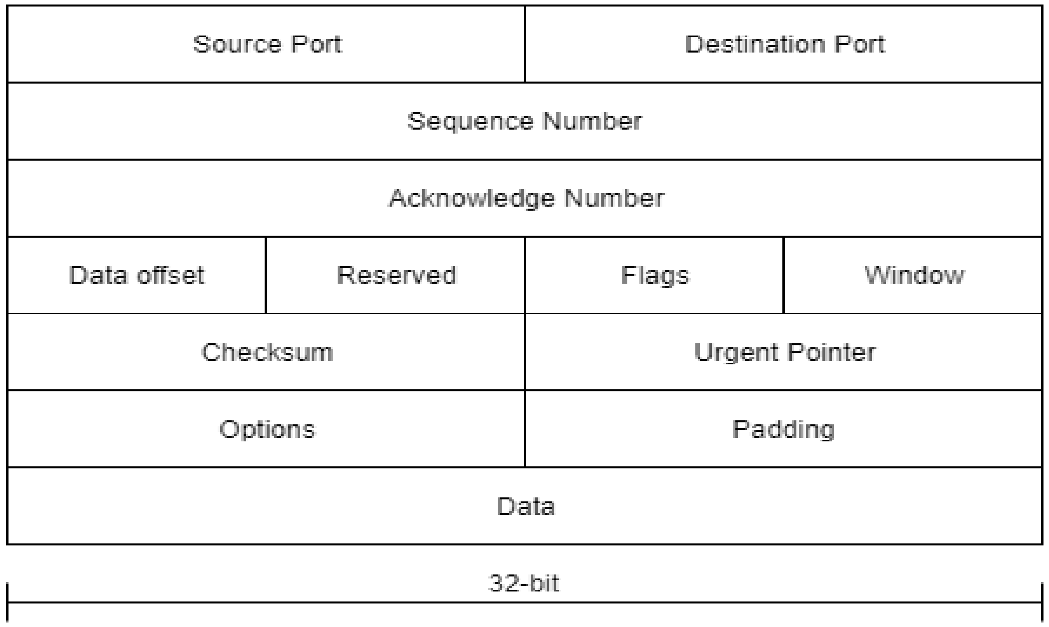


Figure 2.2. TCP packet format

From Figure 2.2, TCP packets consist of following fields:

1. Source and Destination Ports: These are 16- bit each and identify the end points of a TCP connection.
2. Sequence Number: Its size is 32 bits. This field specifies the number assigned to the first byte of the current message. In some conditions, it can also be used to identify the initial sequence number belonging to a future data transmission.

3. Acknowledgement Number: It is also 32-bits. This field contains the value of the next sequence number that the sender of the segment is expected to receive, if the corresponding flag of this field is set. It must be understood that the sequence number corresponds to the data stream in the same direction as the segment, while the acknowledge number refers to the opposite direction as of the segment.
4. Data Offset: This field is also known as header length. Its size is variable, and it conveys the information about the number of words in the TCP header whose size is 32 bits.
5. Reserved bit: It is for future use. The default value of this bit is zero.
6. Flags: The flags convey information about the status of the TCP connection. Each of the 1-bit flags are:
    i. URG: conveys important data is in the segment.
    ii. ACK: indicates the acknowledgement number used is valid.
    iii. PSH: pushes for data to be passed to the application as early as possible.
    iv. RST: used to reset the connection.
    v. SYN: used to synchronize the sequence numbers used for initiation of a connection.
    vi. FIN: indicates that the sender of has finished sending the data and that the connection can be terminated now.
7. Window: This field has 16 bits and conveys information about the receive window of the sender. In other words, it conveys how much buffer is available for incoming data.
8. Checksum: The size is 16 bits and helps error recovery by conveying whether or not the header was damaged during transit.
9. Urgent Pointer: It’s 16 bits and points to the first urgent data byte in the TCP packet.

10. Options: This field is of variable length; and specifies TCP options such as the Maximum Segment Size(MSS) conveying information about the largest block of data that the sender would send to the receiver over the TCP; or the window scale factor allowing for the window size to go up for lines with high bandwidth [13].

11. Padding: This consists of zeros whose number is determined by the size of the Options field. Padding ensures that the header boundary is at 32 bits and data begins right after.

12. Data: This contains upper layer information. This is also variable-sized [11].

TCP has a three-way handshake to establish connection between a client and server or between two clients. Before a client attempts to establish connection with a server, the server needs to first bind to and listen at a port to make it available for connections, called as the passive open [11]. Once the passive open is established, a client is able to initiate an active open. This process involves three steps and both the ends are required to exchange the synchronization (SYN) and acknowledgment (ACK) packets before the real data communication process starts. This is the reason the three-way handshake is also called the syn-synack-ack handshake. The handshake is designed in a way such that both ends help to initiate, negotiate, and separate TCP socket connections at the same time. It allows transfer of multiple TCP socket connections in both directions simultaneously[16]. The process is described in Figure 2.3 below:

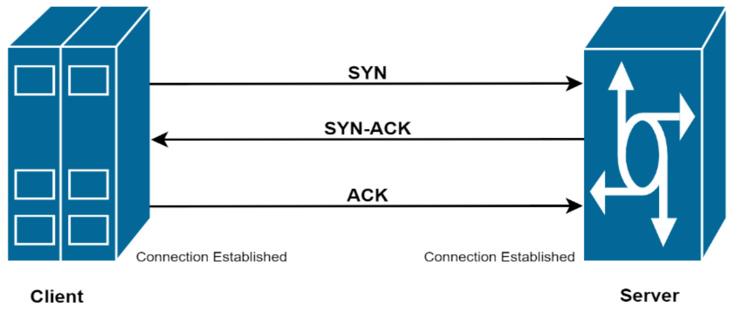


Fig. 2.3. TCP three-way handshake for connection setup

The steps of establishing the connection are as follows:

Step 1. The client establishes attempts to establish a connection with the server by sending a segment with synchronization (SYN) request. The client sets the segment's sequence number to a random value, X.

Step 2. In response, the server sends back a segment with an acknowledgement for the synchronization segment sent by the client in the previous step – SYN-ACK.

The acknowledgment number is set to one number ahead of the received sequence number, i.e. to X + 1; and the sequence number that the server chooses for the packets is another random number, Y.

Step 3. Finally, the client responds back with an acknowledgment (ACK) of the received response from the server in Step 2. This is called the Forward Acknowledgement. The sequence number gets set to the acknowledgment value received in step 2, i.e. X+1; and the acknowledgement number gets set to one value greater than the received sequence number in Step 1, i.e. Y+1[11].

When both the client and server acknowledge each other’s requests, the handshake is successful and a dedicated connection is established between the two machines[17]. After this the actual data gets transferred and the connection stays open until data exchange is complete.

The completion of the data exchange is indicated by exchange of FIN flags and their corresponding acknowledgements from both the ends. The process is shown in Figure 2.3.

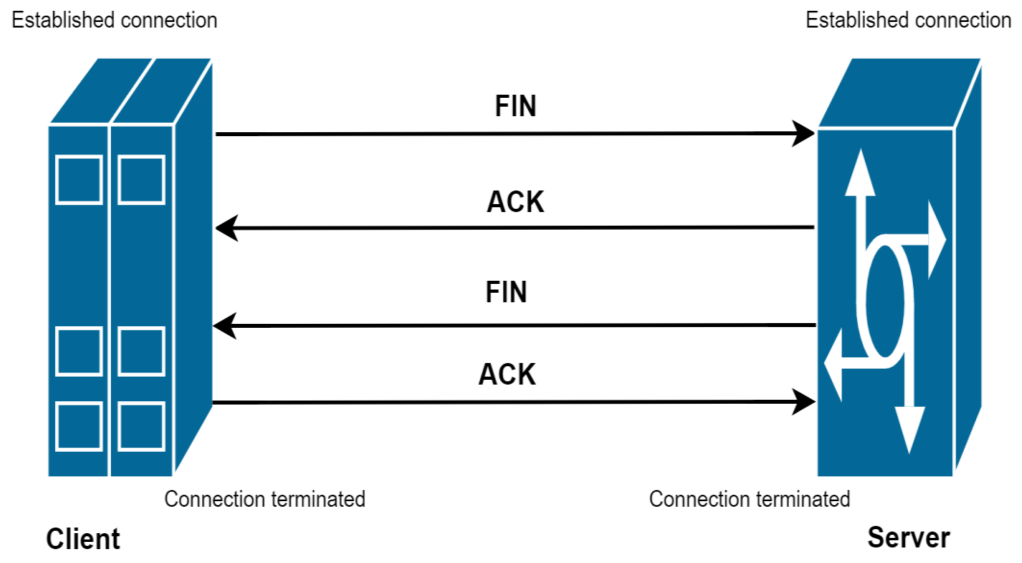


Fig. 2.4. TCP connection termination

It involves the following:

Step 1. The client sends the segment with FIN flag set as high indicating the completion of data transmission.

Step 2. The server replies with an ACK confirming that it received the FIN from the client.

Step 3. When server has no further data to send to the client, it sends a FIN segment to the latter.

Step 4. The client side sends back the ACK to the server side conveying it received the FIN from the latter as well. Now the data exchange has completed from both the ends and the connection terminates.

2.4 Agreement In Scrybe

To guarantee *Scrybe* reaches a consensus, we use Lamport's Byzantine- fault-tolerant (BFT) algorithm[28]. BFT can tolerate less than $\frac{1}{3} * n$ faulty nodes (f), where n is the total number of participating miners in each round. BFT has been employed by a large number of researchers in their work to tolerate faulty nodes. [[18],[19],[20],[21]].

2.5 Related Work

There is an extensive body of work on the D/DoS attacks and their implications. Liang *et al*.[22] implemented a DoS attack method for an IoT system using IP Spoofing and provided results which of the SYN flood attack was the most effective against the security of the IoT technology. The work included evaluation of CPU and memory usage change during the attack and the effect of the size of attack packets on the impact of the DoS attack.

Kshirsagar *et al.*[23] implemented a LAND DoS attack using IP spoofing and conducted a performance analysis of the LAND DoS detection comparing their proposed Intrusion Detection System [IDS] with state-of-the-art systems. Their proposed architecture consists of network traffic analyzer, feature identification and extraction.

Vasek *et al.*[24] provide an empirical insight into impact of DDoS attacks on operators of distributed networks such as the Bitcoin, stating that services with anti-DDoS protection are three times more likely to be attacked.

Johns Hopkins -APL provided an elaborate discussion on DDoS Tradeoff Analysis. They discuss how the SYN flood attacks monopolize the resources, the relations between the attacks, mitigation techniques and the attacker performance such that ways can be found to lower the attacker influences[25].

Congcong *et al.*[23] conducted a security analysis of blockchain with a case study of 51 % attacks. They have simulated the blockchain process and deduced the rule between the blockchain security, attacking and attacking method and attacking power. Their study found how by adjusting the attacking power, most of the blockchain states can be found and the probability of the honest state becoming the attacking state can be analyzed.

K. Geetha *et al.*[24] have conducted identification and analysis of SYN floods against an Ad Hoc network. They implement the SYN flood using IP Spoofing and analyze how the Quality of Service (QoS) parameters such as the packet ratio, average end-to-end delay change due the attack.Their findings show how the legitimate users experience a lower QoS due to the attack.

Saket *et al.*[25] conducted a survey of DDoS attacks on the TCP/IP protocol vulnerabilities. They discuss how the attacks could drain the computational resources of the victim within a short span of time. They also discuss and compare different attack tools such as LOIC, MStream and Switchblade in terms of the vulnerabilities they exploit.

Al-Hawawreh[26] has worked on syn-flood attack on a virtual cloud and its detection based on TCP-IP header statistical features using machine learning techniques. Their findings include the CPU utilization comparison between before the attack and during the attack scenarios.

Kshirsagar *et al.*[28] have discussed the CPU load utilization and minimization for TCP syn floods. Their work includes a survey of the attack detection techniques and a proposed attack detection method based on the CPU threshold load and misuse detection. The analysis is primarily based on how the TCP syn floods cause a abrupt surge in the CPU load.

CHAPTER 3

Security Attribute Verification Using DDoS Tests

Once a distributed system has been designed to maintain performability, it becomes important to verify that the system is successful. To do this, we consider common approaches for disabling, or degrading distributed systems. The most successful tools are generally referred to as Distributed Denial of Service (DDoS) attacks [16]. To verify our proposed blockchain network, we therefore integrate successful DDoS methods into our verification suite. Since *Scrybe* uses TCP for communication, we leverage DoS attacks as a tool to exploit the known flaws in the TCP such as the SYN flood D/DoS attack.

In Section 3.1 we describe the experiment platform; in Section 3.2, we describe the experiment design, Sections 3.2.1, 3.2.2, and 3.2.3 describe the TCP vulnerabilities – SYN flood with IP spoofing, SYN flood with Randomized IP spoofing and the LAND attack respectively- we exploit to verify the system robustness.

3.1 Experiment Platform

The proposed network is evaluated for performability in presence of SYN flood with IP spoofing for both the cases of reachable and unreachable spoofed node, SYN flood with randomized IP Spoofing and LAND attack. We evaluate the response time of the network when a new miner attempts to join the existing network by sending a blockchain request. This is the most strenuous time for the network because the entire length of blockchain mined so far is sent to the requesting miner in this step. If the network handles this well, we assume it is robust in presence of DoS attacks. In addition to this, we try to optimize the kernel parameters to improve the utility of syn cookies against the SYN flood attacks.

The components of the network are:

1. **Miners**: There are *N* participating miners. Miners receive transactions from Clients. The miner selected using the algorithm in Section 2.1 produces the new block, broadcasts it to all the miners. Then each miner appends it to the last block of the blockchain locally.
2. **Client:** Clients submit transactions to the miners to secure their data, and make it immutable. For our experiment, we configure a single client that submits transactions to all miners. This is sufficient for these test scenarios, since the process is symmetric for all miners.
3. **Attacking node**: The attacking node can send attack traffic to any miner. This node can be either a malicious miner or an external node.
4. **Hping3**: is the tool we used to generate attack traffic. It is widely used for testing. It is versatile and robust.
5. **Wireshark**: is the network analyzer we use.

3.2 The Experiment Design

We configure seven Ubuntu 18.04.2 virtual machines (VMs) on seven distinct physical host machines in our Network Security Lab. The specifications of each are provided in Table 1.

Table I : Machine specifications

Memory	2 Gb
Storage (Hard Disk)	10 Gb
Number of CPUs	1

We configure five VMs as miners and one as a client. The client submits one xml file every 3 seconds to all participating miners. One of the five miners is assumed to be malicious. The malicious miner VM sends attack traffic using hping3 to one of the honest miners. A new miner attempts to join the network using IP address of any of the existing miners. and requests a blockchain update. One miner is randomly selected to send the blockchain update. Wireshark captures traffic on the victim miner. The blockchain update is required to know the list of all the participating miners; and the length of blockchain produced so far. Unless the new miner has the blockchain produced so far, it can't produce the new block and broadcast to the rest of the miners if it gets selected by the LWM algorithm to add the new block. This is a very important step in the implementation and is strenuous in terms of computation. Therefore, if our network performs with an acceptable level of robustness in this step, we assume it can handle all other steps. We use the time taken by the new node to have its request processed to measure *Scrybe's* resilience and robustness. The IP addresses of the nodes are in Table II.

Table II :IP Addresses Used

Device	**Virtual Machine**	**IP Address**
Personal Computer 1	Scrybe 1	192.168.10.108
Personal Computer 2	Scrybe 2	192.168.10.112
Personal Computer 3	Scrybe 3	192.168.10.150
Personal Computer 4	Scrybe 4	192.168.10.134
Personal Computer 5	Scrybe 5	192.168.10.138
Personal Computer 6	Scrybe 6	192.168.10.136
Personal Computer 7	Scrybe 7	192.168.10.110

We evaluate system performance for the attacks below. SYN cookies are enabled for each mining node throughout our experiments. When a malicious miner launches the attack, care is taken to maintain a ratio of less than $\frac{1}{3} * N$ malicious miners to honest miners. N is the total number of miners. For our evaluation, we use N = 5.

3.2.1 Syn Flood With Ip Spoofing:

SYN flood attacks exploit the TCP three-way handshake. The target is to have as many half-open TCP connections as possible and slow down the victim. In a normal TCP connection, a client sends a SYN request. The server responds with a SYN-ACK acknowledgment packet. After receiving the SYN-ACK from the server, the client responds with an ACK packet[12]. Then the connection is established and data is exchanged. However, to establish an attack, the attacker sends an enormous number of SYN requests to the server. The server can't differentiate between the SYN request of a malicious node and a legitimate SYN request. Therefore, it attempts to send a SYN-ACK in response to incoming SYN requests. However, the TCP queue of the server fills due to the large flood of SYN packets rendering it unable to handle more incoming requests for a TCP connection until the queue is reaped out. Any service requested from the server during this time experiences a denial-of-service.

IP Spoofing uses a forged IP address as the source IP of an IP packet. The node sending the packet fools the host with the target IP address into sending its reply packet to the node with the forged address. This helps attackers escape detection.

This can be illustrated by:

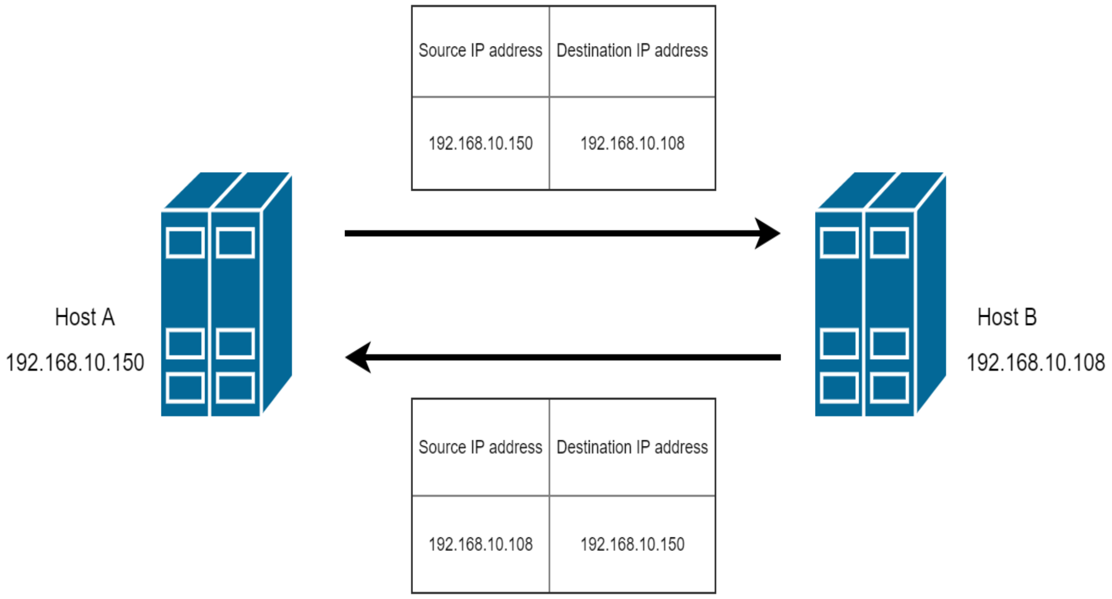


Figure 3.1(a). Normal traffic network

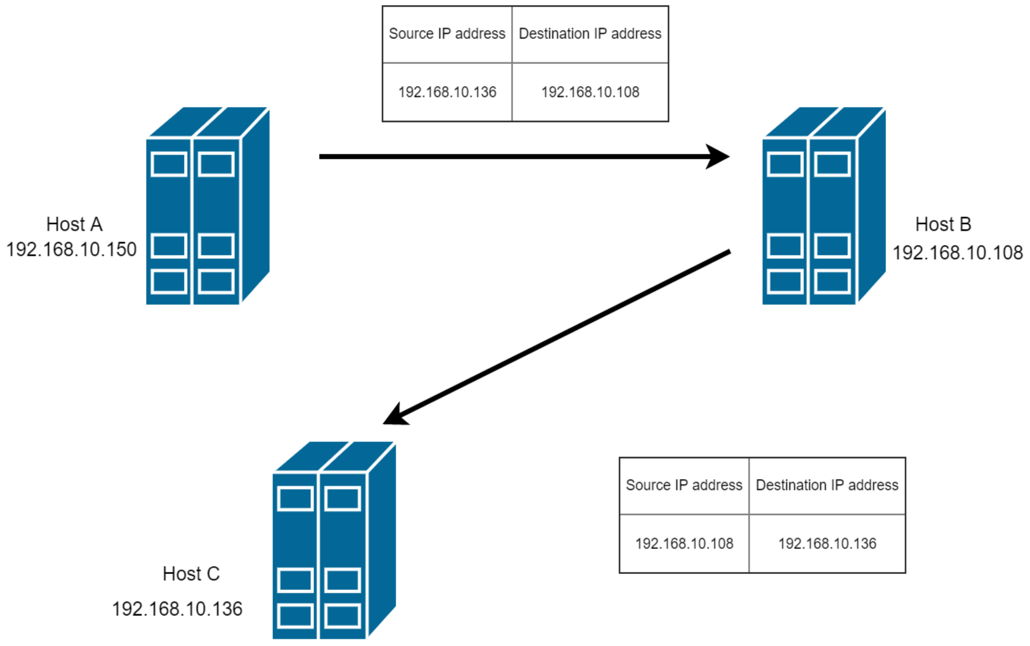


Figure 3.1(b). Network traffic with spoofed IP address

In Figure 3.1(a), the network traffic operates normally. Host A sends traffic using its own IP address and destination Host B replies to Host A. In Figure 3.1(b), Host A forges the IP address from Host C while sending traffic to Host B. Consequently, Host B sends replies to Host C.

This is exploited by attackers by using the IP address of the target (victim) as the source address while sending traffic requests to an enormous number of machines and directing the reply packets to the target. Consequently, the target experiences resource throttling and authorized users of the applications hosted on Host C experience a denial or delay of service.

3.2.1.1 Why Ip Spoofing Works:

IP routing follows a hop by hop mechanism. Each packet is routed separately, and the route of each packet is decided by the routers the packets passes through. The reason why IP spoofing is possible is because routers only use the destination IP address in the packet header to make routing decisions. This means whether or not the source IP is valid, the packet delivery won't be affected. The destination only uses that address to reply to the source [28].

IP Spoofing can be accomplished primarily in the following two ways :

A. With Spoofed Host Reachable

For evaluating the performance of *Scrybe* under the influence of this attack, we configure the VMs as follows:

Table III: Nodes and their role for IP spoofing with spoofed host reachable

IP Address	Role
192.168.10.150	Miner (victim)
192.168.10.134	Miner (spoofed)
192.168.10.110	Miner
192.168.10.108	Miner
192.168.10.136	New Miner requesting update
192.168.10.138	Client
192.168.10.112	Malicious miner [Attacking node]

Implementation:

We first use hping3 on the attacking node to see if the port we want to attack on is listening or not.

The command used is: **hping3 -V -p 10987 -S -c 1 192.168.10.150** ; where *V* means verbose mode, *10987* is the port where our miners communicate with other nodes for our blockchain network and hence is our target port, *S* means we are sending SYN traffic[9], *c 1* means we stop after sending one packet and receiving its reply packet of 1 and *192.168.10.150* is our target machine.

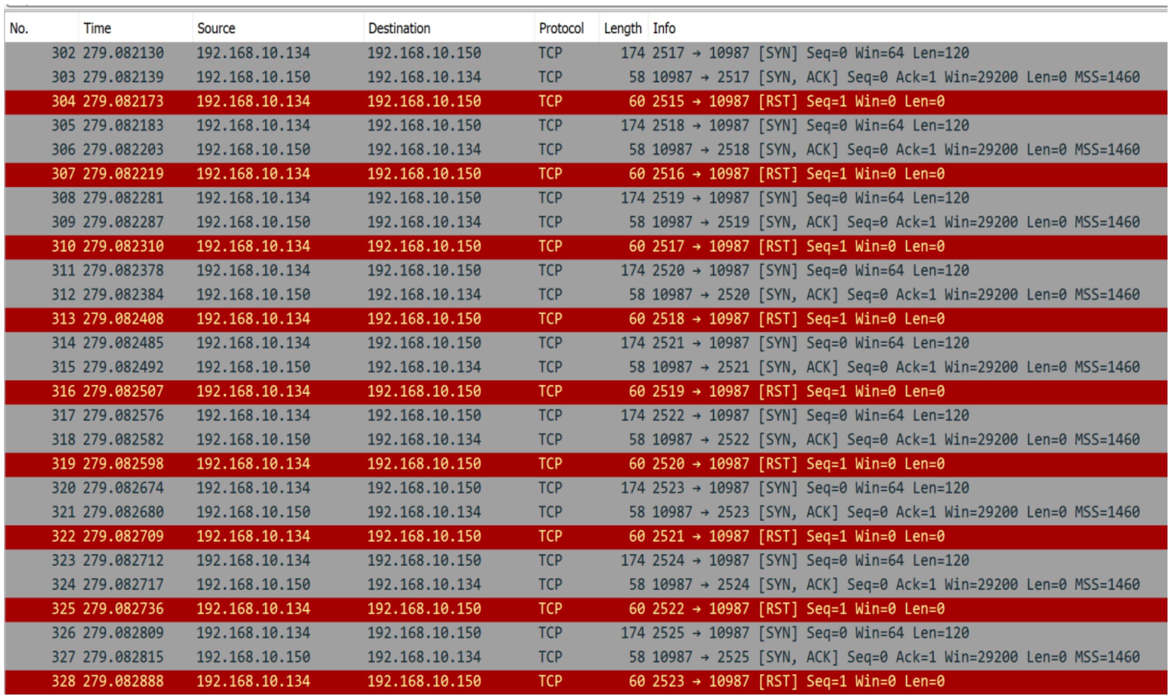
After receiving a SYN-ACK packet in return, we confirm that the port is open and we can successfully send attack traffic to it. Next, while the initial miners are mining and the client is submitting files, we launch attack on the target miner using hping3 on the attacking node. To demonstrate detection-evasion and attempt to malign an honest participating miner, we spoofed the IP address of the latter while launching the attack. The new miner asks for a blockchain update while the attack is ongoing.

The command used for the attack is: **hping3 -V -p 10987 -S -d 120 –flood 192.168.10.150 -a 192.168.10.134;** where *flood* continuously sends packets as quickly as possible without showing replies , *-a* is used to spoof IP addresses while attacking, which, in our case is *192.168.10.134*and *-d* lets us set the data packet size in bytes, which in our case is *120.*

Another relevant parameter relevant is TCP window size. We use the default value of 64 in hping3 through out our experiments. Therefore, we don’t need to configure that explicitly in the script. Since we are configuring the attack in *flood* mode, we don’t configure the count *c* because that would be overridden by the *flood* mode.

A portion Wireshark trace from the experiment is in Figure 3.2. SYN requests and SYN-ACK responses are shown. No final ACK from the client complete the handshake. We used the following command on Wireshark to filter this traffic:

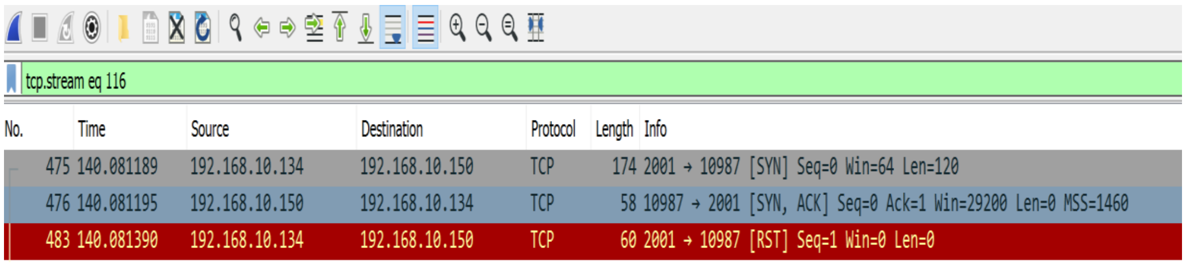
tcp.flags.syn==1 && (tcp.flags.syn==0



```
No.  Time        Source          Destination     Protocol Length Info
 302 279.082130  192.168.10.134  192.168.10.150  TCP       174 2517 → 10987 [SYN] Seq=0 Win=64 Len=120
 303 279.082139  192.168.10.150  192.168.10.134  TCP        58 10987 → 2517 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 304 279.082173  192.168.10.134  192.168.10.150  TCP        60 2515 → 10987 [RST] Seq=1 Win=0 Len=0
 305 279.082183  192.168.10.134  192.168.10.150  TCP       174 2518 → 10987 [SYN] Seq=0 Win=64 Len=120
 306 279.082203  192.168.10.150  192.168.10.134  TCP        58 10987 → 2518 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 307 279.082219  192.168.10.134  192.168.10.150  TCP        60 2516 → 10987 [RST] Seq=1 Win=0 Len=0
 308 279.082281  192.168.10.134  192.168.10.150  TCP       174 2519 → 10987 [SYN] Seq=0 Win=64 Len=120
 309 279.082287  192.168.10.150  192.168.10.134  TCP        58 10987 → 2519 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 310 279.082310  192.168.10.134  192.168.10.150  TCP        60 2517 → 10987 [RST] Seq=1 Win=0 Len=0
 311 279.082378  192.168.10.134  192.168.10.150  TCP       174 2520 → 10987 [SYN] Seq=0 Win=64 Len=120
 312 279.082384  192.168.10.150  192.168.10.134  TCP        58 10987 → 2520 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 313 279.082408  192.168.10.134  192.168.10.150  TCP        60 2518 → 10987 [RST] Seq=1 Win=0 Len=0
 314 279.082485  192.168.10.134  192.168.10.150  TCP       174 2521 → 10987 [SYN] Seq=0 Win=64 Len=120
 315 279.082492  192.168.10.150  192.168.10.134  TCP        58 10987 → 2521 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 316 279.082507  192.168.10.134  192.168.10.150  TCP        60 2519 → 10987 [RST] Seq=1 Win=0 Len=0
 317 279.082576  192.168.10.134  192.168.10.150  TCP       174 2522 → 10987 [SYN] Seq=0 Win=64 Len=120
 318 279.082582  192.168.10.150  192.168.10.134  TCP        58 10987 → 2522 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 319 279.082598  192.168.10.134  192.168.10.150  TCP        60 2520 → 10987 [RST] Seq=1 Win=0 Len=0
 320 279.082674  192.168.10.134  192.168.10.150  TCP       174 2523 → 10987 [SYN] Seq=0 Win=64 Len=120
 321 279.082680  192.168.10.150  192.168.10.134  TCP        58 10987 → 2523 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 322 279.082709  192.168.10.134  192.168.10.150  TCP        60 2521 → 10987 [RST] Seq=1 Win=0 Len=0
 323 279.082712  192.168.10.134  192.168.10.150  TCP       174 2524 → 10987 [SYN] Seq=0 Win=64 Len=120
 324 279.082717  192.168.10.150  192.168.10.134  TCP        58 10987 → 2524 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 325 279.082736  192.168.10.134  192.168.10.150  TCP        60 2522 → 10987 [RST] Seq=1 Win=0 Len=0
 326 279.082809  192.168.10.134  192.168.10.150  TCP       174 2525 → 10987 [SYN] Seq=0 Win=64 Len=120
 327 279.082815  192.168.10.150  192.168.10.134  TCP        58 10987 → 2525 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 328 279.082888  192.168.10.134  192.168.10.150  TCP        60 2523 → 10987 [RST] Seq=1 Win=0 Len=0
```

Figure 3.2. A portion of attack traffic captured on the victim node during the attack using Wireshark for IP spoofing with spoofed host reachable

A portion of the TCP stream of the attack packets is shown in Figure 3.3.



```
tcp.stream eq 116

No.  Time        Source          Destination     Protocol Length Info
 475 140.081189  192.168.10.134  192.168.10.150  TCP       174 2001 → 10987 [SYN] Seq=0 Win=64 Len=120
 476 140.081195  192.168.10.150  192.168.10.134  TCP        58 10987 → 2001 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
 483 140.081390  192.168.10.134  192.168.10.150  TCP        60 2001 → 10987 [RST] Seq=1 Win=0 Len=0
```

Figure 3.3: TCP streams of the attack packets for IP spoofing with spoofed host reachable

Observations:

From Figure 3.2, we see a large number of SYN packets being sent continuously to the victim. The packets appear to have been sent from miner with IP address as 192.168.10.134, however, those are actually launched by the malicious miner with IP address as 192.168.10.112. The server sends back the SYN-ACK reply to each of the SYN requests from 192.168.10.134 since it has no way to tell those from non-malicious packets. Since the machine with IP address 192.168.10.134 didn't initiate any of those SYN connections, it responds with an RST packet in return, thereby terminating the connection.

From Figure 3.3, we observe the TCP stream for one of the attack packet. Similar TCP stream was observed for all the attack packets. It appears that the spoofed machine responded with a RST packet to the SYN-ACK reply the target machine sent to it because it thought that was the source IP address of the initial SYN packet.

B. With Spoofed Host Unreachable

It is also possible that the attacking nodes spoof the IP of a machine which has been powered off. This avoids the spoofed machine terminating the connection. The VMs are configured as follows:

TABLE IV: Nodes and their role for IP spoofing with spoofed host unreachable

IP Address	Role
192.168.10.150	Miner (victim)
192.168.10.134	Spoofed [Switched off]
192.168.10.110	Miner
192.168.10.108	Miner
192.168.10.136	New Miner requesting update
192.168.10.138	Client
192.168.10.112	Malicious miner [Attacking node]

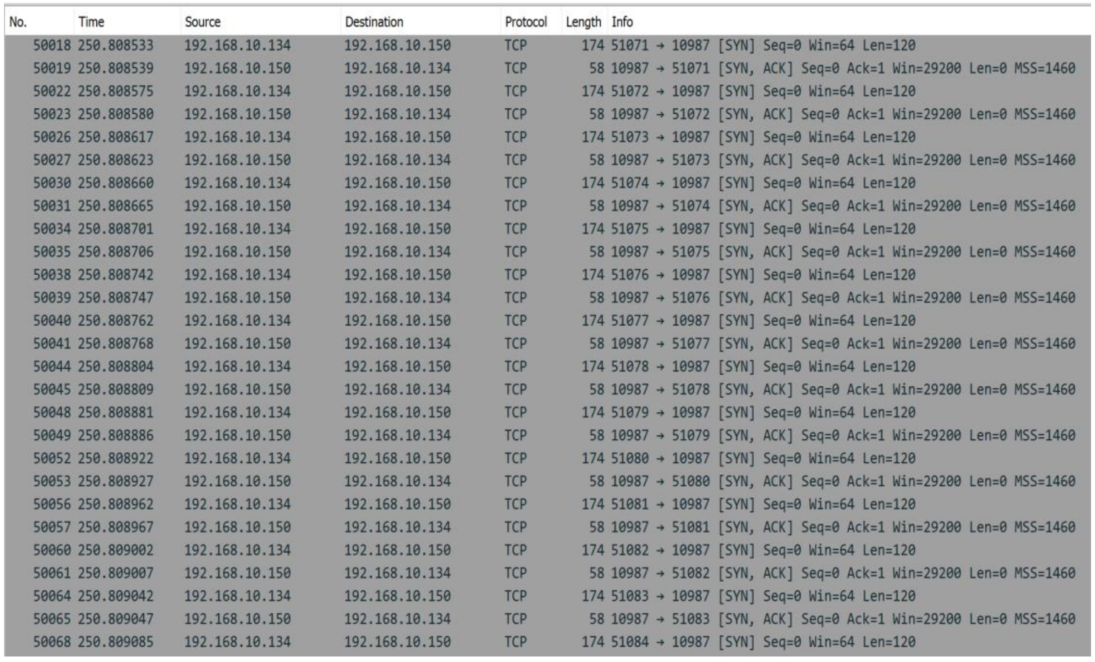
Implementation:

The implementation of this case is similar to section A. The only difference is that the spoofed IP belongs to a VM which has been turned off. While the initial miners are mining and the client is submitting files, we launch attack on the target miner using hping3 on the attacking node. The new miner asks for a blockchain update while the attack is ongoing.

The command used for the attack is : **hping3 -V -p 10987 -S -d 120 - - flood 192.168.10.150 -a 192.168.10.134** *;* where *192.168.10.150* is the victim miner and *-a* is used to spoof the IP of other node while attacking, which, in our case is *192.168.10.134*. VM with IP address 192.168.10.134 is switched off.

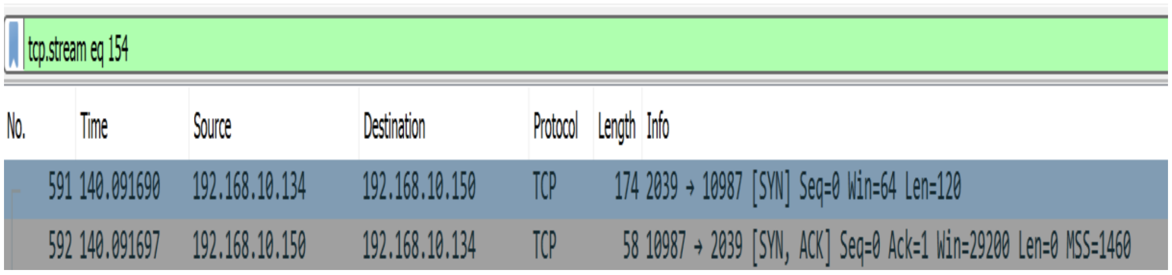
A portion of the captured traffic on Wireshark is shown in Figure 3.4. However, the TCP stream is different, as shown in Figure 3.5.

tcp.flags.syn==1 && tcp.flags.ack==0



```
No.      Time        Source           Destination      Protocol  Length Info
   50018 250.808533  192.168.10.134   192.168.10.150   TCP       174 51071 → 10987 [SYN] Seq=0 Win=64 Len=120
   50019 250.808539  192.168.10.150   192.168.10.134   TCP        58 10987 → 51071 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50022 250.808575  192.168.10.134   192.168.10.150   TCP       174 51072 → 10987 [SYN] Seq=0 Win=64 Len=120
   50023 250.808580  192.168.10.150   192.168.10.134   TCP        58 10987 → 51072 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50026 250.808617  192.168.10.134   192.168.10.150   TCP       174 51073 → 10987 [SYN] Seq=0 Win=64 Len=120
   50027 250.808623  192.168.10.150   192.168.10.134   TCP        58 10987 → 51073 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50030 250.808660  192.168.10.134   192.168.10.150   TCP       174 51074 → 10987 [SYN] Seq=0 Win=64 Len=120
   50031 250.808665  192.168.10.150   192.168.10.134   TCP        58 10987 → 51074 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50034 250.808701  192.168.10.134   192.168.10.150   TCP       174 51075 → 10987 [SYN] Seq=0 Win=64 Len=120
   50035 250.808706  192.168.10.150   192.168.10.134   TCP        58 10987 → 51075 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50038 250.808742  192.168.10.134   192.168.10.150   TCP       174 51076 → 10987 [SYN] Seq=0 Win=64 Len=120
   50039 250.808747  192.168.10.150   192.168.10.134   TCP        58 10987 → 51076 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50040 250.808762  192.168.10.134   192.168.10.150   TCP       174 51077 → 10987 [SYN] Seq=0 Win=64 Len=120
   50041 250.808768  192.168.10.150   192.168.10.134   TCP        58 10987 → 51077 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50044 250.808804  192.168.10.134   192.168.10.150   TCP       174 51078 → 10987 [SYN] Seq=0 Win=64 Len=120
   50045 250.808809  192.168.10.150   192.168.10.134   TCP        58 10987 → 51078 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50048 250.808881  192.168.10.134   192.168.10.150   TCP       174 51079 → 10987 [SYN] Seq=0 Win=64 Len=120
   50049 250.808886  192.168.10.150   192.168.10.134   TCP        58 10987 → 51079 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50052 250.808922  192.168.10.134   192.168.10.150   TCP       174 51080 → 10987 [SYN] Seq=0 Win=64 Len=120
   50053 250.808927  192.168.10.150   192.168.10.134   TCP        58 10987 → 51080 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50056 250.808962  192.168.10.134   192.168.10.150   TCP       174 51081 → 10987 [SYN] Seq=0 Win=64 Len=120
   50057 250.808967  192.168.10.150   192.168.10.134   TCP        58 10987 → 51081 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50060 250.809002  192.168.10.134   192.168.10.150   TCP       174 51082 → 10987 [SYN] Seq=0 Win=64 Len=120
   50061 250.809007  192.168.10.150   192.168.10.134   TCP        58 10987 → 51082 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50064 250.809042  192.168.10.134   192.168.10.150   TCP       174 51083 → 10987 [SYN] Seq=0 Win=64 Len=120
   50065 250.809047  192.168.10.150   192.168.10.134   TCP        58 10987 → 51083 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
   50068 250.809085  192.168.10.134   192.168.10.150   TCP       174 51084 → 10987 [SYN] Seq=0 Win=64 Len=120
```

Figure 3.4. A portion of attack traffic captured on the victim node during the attack using Wireshark for IP spoofing with spoofed host unreachable



```
tcp.stream eq 154
No.   Time          Source           Destination      Protocol  Length  Info
  591 140.091690    192.168.10.134   192.168.10.150   TCP          174  2039 → 10987 [SYN] Seq=0 Win=64 Len=120
  592 140.091697    192.168.10.150   192.168.10.134   TCP           58  10987 → 2039 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
```

Figure 3.5. TCP streams of one of the attack packets for IP spoofing with spoofed host unreachable

Observations

From Figure 3.4, we observe that unlike in Figure 3.2, there is no RST packet sent back from the spoofed IP machine because the IP belongs to a machine which is inactive(switched off).There is a large number of incomplete TCP connections because the initiator of the SYN handshake never sent the final ACK packet required to establish the connection. Instead of that, there is a large inflow of new SYN requests from the same IP address.

Figure 3.5 has TCP stream of one of the attack packets to further establish that the TCP connection is half-open and just using the resources of the victim. Each attack packet has similar TCP stream.

3.2.2 Syn Flood With Randomized Ip Spoofing

IP Spoofing with a single source IP spoofed is comparatively less strong to escape detection and foiling. Using botnets is another way of launching DDoS attacks. Botnets are an internet-connected network of devices which are running one or more bots, where a bot is a malware-infected compromised machine that can remotely be controlled by a cybercriminal. However, cybercriminals wishing to launch DDoS attacks but also wanting to avoid using large botnets can send flood traffic with randomized IP spoofing. This helps them to conceal their own identity as well as makes blocking the attack more challenging as it appears to be originating from many sources simultaneously[28].

For evaluating the performance of *Scrybe* under the influence of this attack, we configure the VMs as follows:

TABLE V: Nodes and their role for randomized IP Spoofing attack

IP Address	Role
192.168.10.136	Miner (victim)
192.168.10.134	Miner
192.168.10.110	Miner
192.168.10.108	Miner
192.168.10.150	New Miner requesting update
192.168.10.138	Client
192.168.10.112	Malicious miner [Attacking node]

Implementation:

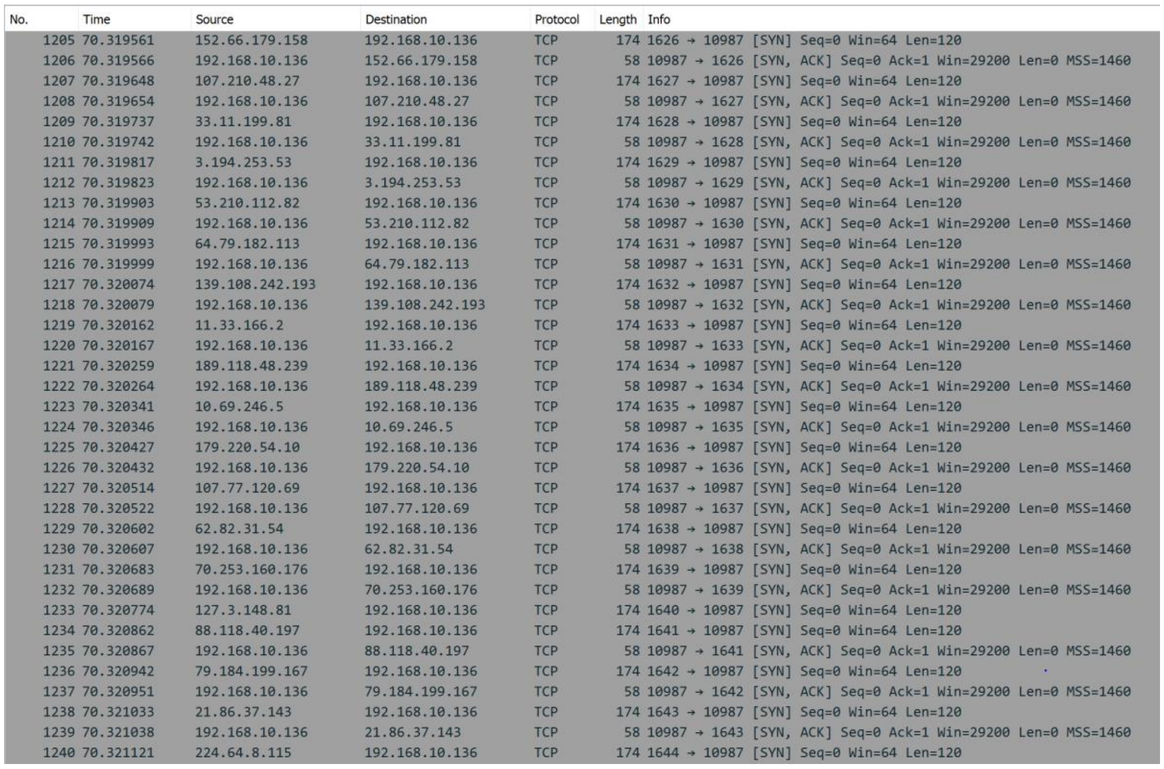
The implementation of this case is similar to section A. The difference is that the attack traffic comes from a large number of random IP addresses available on the internet for this. While the initial miners are mining and the client is submitting files, we launch attack on the target miner using hping3 on the attacking node keeping source node as random. The new miner asks for a blockchain update while the attack is ongoing.

The command used is: **hping3 -V -d 120 -S -p 10987 - -flood - - rand source 192.168.10.136** *;*where *192.168.10.136* is the victim miner and *rand source* is the mode of the sending the attack traffic. In this mode, hping3 looks for random IP addresses available over the internet and makes the attack seem to be launched from all those. This option stresses the firewall state tables and other per-ip dynamic tables (which can be used to block traffic from a particular malicious IP address) in the TCP/IP stack and the firewall software.

The captured traffic on Wireshark is shown in Figure 3.6; and the TCP stream is different, as shown in Figure 3.7.

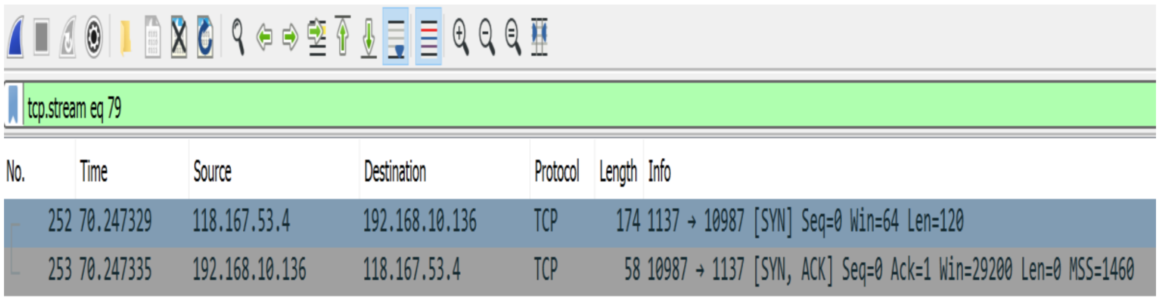
As mentioned in previous sections, the Wireshark filter used to see the attack traffic is :

tcp.flags.syn==1 && tcp.flags.ack==0



```
No.     Time       Source           Destination      Protocol  Length  Info
  1205 70.319561  152.66.179.158   192.168.10.136   TCP        174 1626 → 10987 [SYN] Seq=0 Win=64 Len=120
  1206 70.319566  192.168.10.136   152.66.179.158   TCP         58 10987 → 1626 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1207 70.319648  107.210.48.27    192.168.10.136   TCP        174 1627 → 10987 [SYN] Seq=0 Win=64 Len=120
  1208 70.319654  192.168.10.136   107.210.48.27    TCP         58 10987 → 1627 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1209 70.319737  33.11.199.81     192.168.10.136   TCP        174 1628 → 10987 [SYN] Seq=0 Win=64 Len=120
  1210 70.319742  192.168.10.136   33.11.199.81     TCP         58 10987 → 1628 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1211 70.319817  3.194.253.53     192.168.10.136   TCP        174 1629 → 10987 [SYN] Seq=0 Win=64 Len=120
  1212 70.319823  192.168.10.136   3.194.253.53     TCP         58 10987 → 1629 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1213 70.319903  53.210.112.82    192.168.10.136   TCP        174 1630 → 10987 [SYN] Seq=0 Win=64 Len=120
  1214 70.319909  192.168.10.136   53.210.112.82    TCP         58 10987 → 1630 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1215 70.319993  64.79.182.113    192.168.10.136   TCP        174 1631 → 10987 [SYN] Seq=0 Win=64 Len=120
  1216 70.319999  192.168.10.136   64.79.182.113    TCP         58 10987 → 1631 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1217 70.320074  139.108.242.193  192.168.10.136   TCP        174 1632 → 10987 [SYN] Seq=0 Win=64 Len=120
  1218 70.320079  192.168.10.136   139.108.242.193  TCP         58 10987 → 1632 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1219 70.320162  11.33.166.2      192.168.10.136   TCP        174 1633 → 10987 [SYN] Seq=0 Win=64 Len=120
  1220 70.320167  192.168.10.136   11.33.166.2      TCP         58 10987 → 1633 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1221 70.320259  189.118.48.239   192.168.10.136   TCP        174 1634 → 10987 [SYN] Seq=0 Win=64 Len=120
  1222 70.320264  192.168.10.136   189.118.48.239   TCP         58 10987 → 1634 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1223 70.320341  10.69.246.5      192.168.10.136   TCP        174 1635 → 10987 [SYN] Seq=0 Win=64 Len=120
  1224 70.320346  192.168.10.136   10.69.246.5      TCP         58 10987 → 1635 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1225 70.320427  179.220.54.10    192.168.10.136   TCP        174 1636 → 10987 [SYN] Seq=0 Win=64 Len=120
  1226 70.320432  192.168.10.136   179.220.54.10    TCP         58 10987 → 1636 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1227 70.320514  107.77.120.69    192.168.10.136   TCP        174 1637 → 10987 [SYN] Seq=0 Win=64 Len=120
  1228 70.320522  192.168.10.136   107.77.120.69    TCP         58 10987 → 1637 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1229 70.320602  62.82.31.54      192.168.10.136   TCP        174 1638 → 10987 [SYN] Seq=0 Win=64 Len=120
  1230 70.320607  192.168.10.136   62.82.31.54      TCP         58 10987 → 1638 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1231 70.320683  70.253.160.176   192.168.10.136   TCP        174 1639 → 10987 [SYN] Seq=0 Win=64 Len=120
  1232 70.320689  192.168.10.136   70.253.160.176   TCP         58 10987 → 1639 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1233 70.320774  127.3.148.81     192.168.10.136   TCP        174 1640 → 10987 [SYN] Seq=0 Win=64 Len=120
  1234 70.320862  88.118.40.197    192.168.10.136   TCP        174 1641 → 10987 [SYN] Seq=0 Win=64 Len=120
  1235 70.320867  192.168.10.136   88.118.40.197    TCP         58 10987 → 1641 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1236 70.320942  79.184.199.167   192.168.10.136   TCP        174 1642 → 10987 [SYN] Seq=0 Win=64 Len=120
  1237 70.320951  192.168.10.136   79.184.199.167   TCP         58 10987 → 1642 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1238 70.321033  21.86.37.143     192.168.10.136   TCP        174 1643 → 10987 [SYN] Seq=0 Win=64 Len=120
  1239 70.321038  192.168.10.136   21.86.37.143     TCP         58 10987 → 1643 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  1240 70.321121  224.64.8.115     192.168.10.136   TCP        174 1644 → 10987 [SYN] Seq=0 Win=64 Len=120
```

Figure 3.6. A portion of traffic captured on the victim node during the attack using Wireshark for Randomized IP spoofing



```
tcp.stream eq 79

No.   Time        Source          Destination     Protocol  Length  Info
  252 70.247329   118.167.53.4    192.168.10.136  TCP        174 1137 → 10987 [SYN] Seq=0 Win=64 Len=120
  253 70.247335   192.168.10.136  118.167.53.4    TCP         58 10987 → 1137 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
```

Figure 3.7. TCP stream of one of the attack packets for randomized IP spoofing

Observations

From Figure 3.7, we observe that the SYN flood packets appear to be launched from a large number of random IP addresses across the internet. The victim miner responds to each of the SYN request with a SYN-ACK packet since it has no way to tell a legit SYN request from the malicious one because these requests are not from a single IP as the source address.

From Figure 3.7, we observe the TCP stream of one of the attack packets. This is also an incomplete SYN handshake resulting a half-opened TCP connection. All the attack packets have similar TCP streams.

3.2.3 LAND DoS Attack

LAND attack is a DoS attack where the attacker crafts the attack packets in such a way that the source IP address and the port and the destination IP address and the port of the TCP segments is the same[16], i.e. that of the victim's; and the port chosen is one of the open ports on the victim. In a way, it's a variant of IP spoofing where the spoofed IP address is same as the victim's IP address. This attack results in the victim ending up in a continuous loop (lock-up) of sending packets and responses to itself until the system is crashed or the attack is detected [29,30]. Diagrammatically, LAND DoS attack can be represented as shown in Figure 3.8.

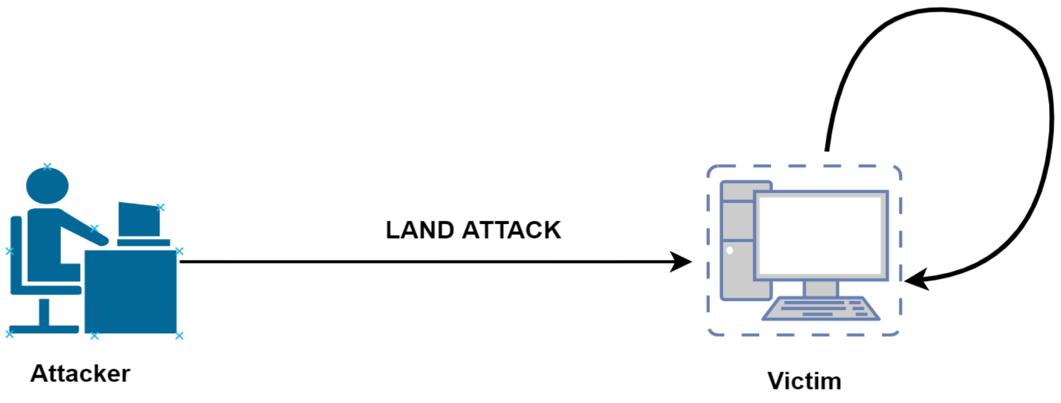


Figure 3.8 . LAND DoS attack illustration

For evaluating the performance of Scrybe under the influence of this attack, we configure the VMs as follows:

TABLE V: Nodes and their roles for LAND attack

IP Address	Role
192.168.10.150	Miner (victim)
192.168.10.110	Miner
192.168.10.108	Miner
192.168.10.136	New Miner requesting update
192.168.10.138	Client
192.168.10.112	Malicious miner [Attacking node]

Implementation:

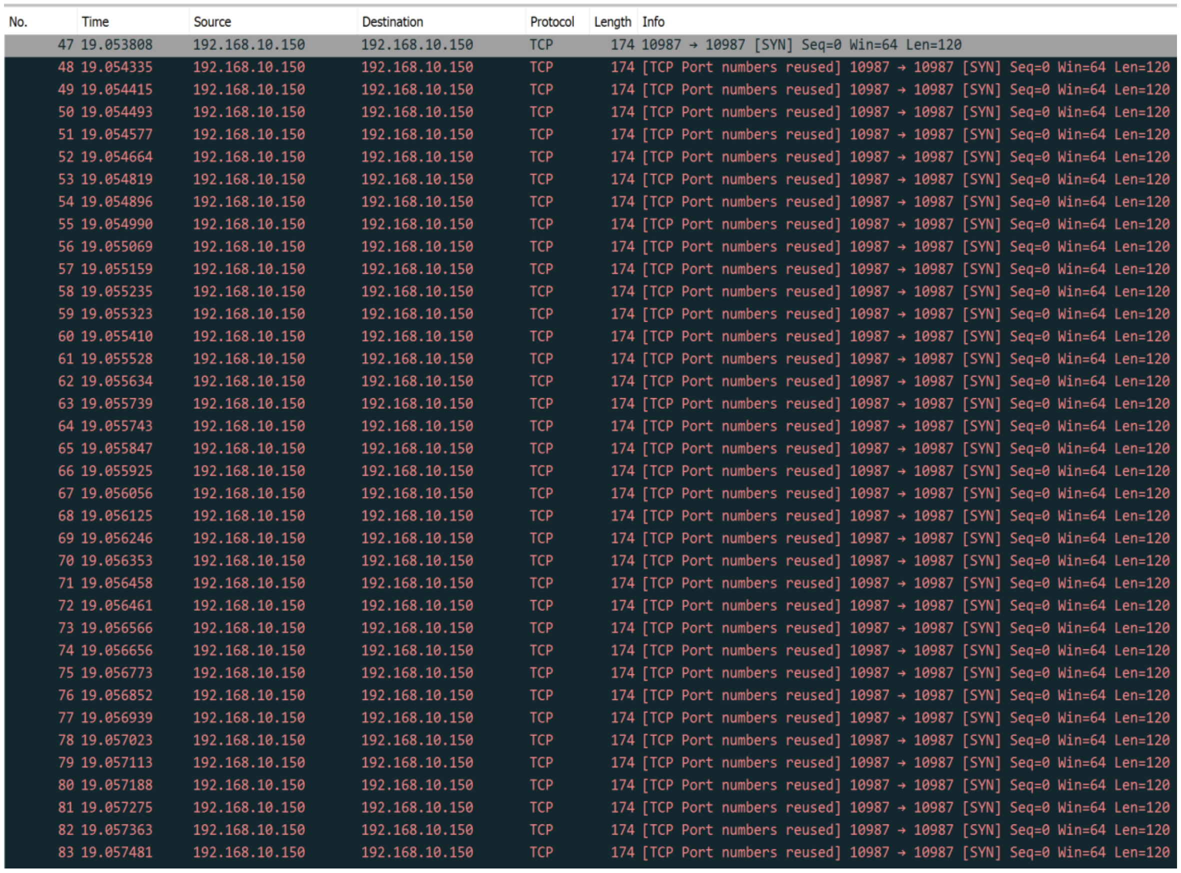
The implementation of this attack makes it seem that a large number of packets are sent from the victim continuously to itself on one of its ports. While the initial miners are mining and the client is submitting files, we launch attack on the target miner using hping3 on the attacking node keeping source node as random. The new miner asks for a blockchain update while the attack is ongoing.

The command used is : **sudo hping3 -V -d 120 -w 64 -S -p 10987 -k -s 10987 - -flood 192.168.10.150 -a 192.168.10.150 ,** where *192.168.10.150* is the victim node, -a is used to spoof the source IP address to the victim's IP address so that the machine keeps sending traffic continuously to itself, -s is used to select a base source port to send the traffic to the destination port, -k is used to keep this port number still, i.e. keep sending traffic from this port continuously. By default, the source port is incremented by one with each new packet sent.

The captured traffic on Wireshark is shown in Figure 3.9; and the TCP stream is different, as shown in Figure 3.10.

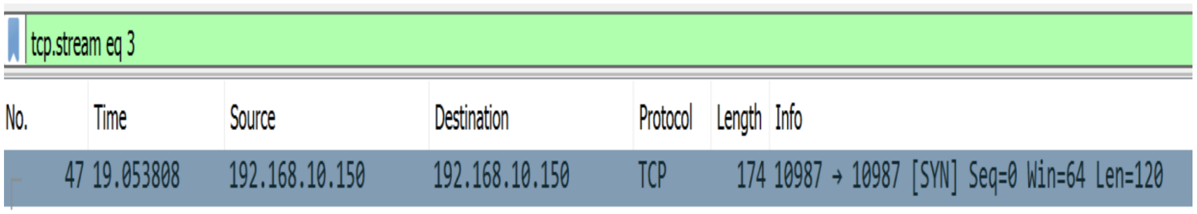
As mentioned in previous sections, the Wireshark filter used to see the attack traffic is :

tcp.flags.syn==1 && tcp.flags.ack==0



```
No.   Time        Source          Destination     Protocol Length Info
   47 19.053808   192.168.10.150  192.168.10.150  TCP      174 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   48 19.054335   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   49 19.054415   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   50 19.054493   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   51 19.054577   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   52 19.054664   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   53 19.054819   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   54 19.054896   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   55 19.054990   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   56 19.055069   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   57 19.055159   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   58 19.055235   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   59 19.055323   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   60 19.055410   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   61 19.055528   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   62 19.055634   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   63 19.055739   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   64 19.055743   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   65 19.055847   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   66 19.055925   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   67 19.056056   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   68 19.056125   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   69 19.056246   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   70 19.056353   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   71 19.056458   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   72 19.056461   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   73 19.056566   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   74 19.056656   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   75 19.056773   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   76 19.056852   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   77 19.056939   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   78 19.057023   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   79 19.057113   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   80 19.057188   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   81 19.057275   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   82 19.057363   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
   83 19.057481   192.168.10.150  192.168.10.150  TCP      174 [TCP Port numbers reused] 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
```

Figure 3.9. A portion of the attack traffic captured on Wireshark for LAND attack



```
tcp.stream eq 3

No.   Time        Source          Destination     Protocol Length Info
   47 19.053808   192.168.10.150  192.168.10.150  TCP      174 10987 → 10987 [SYN] Seq=0 Win=64 Len=120
```

Figure 3.10. TCP stream of one of the attack packets for LAND attack

Observations:

From Figure 3.9, we observe a continuous traffic going from the same IP address to the same destination address, with the same port as the source as the destination port. Since the SYN packet flood is originating from the same source port, i.e. 10987, the Wireshark sees it as an unusual behavior stating the 'TCP Port numbers reused warning'.

Figure 3.10 shows the TCP stream. There is only a SYN packet from source to destination. Due to SYN packets circling around the same port and IP address for both source and destination, the machine drops these and does not reply with a SYN-ACK packet instead.

3.3 Results

In each case of experiments, the mining algorithm runs successfully, and we have a selected miner at the end of each round to produce the new block. The

The response times for each of the attack cases above is noted. For reference, we also checked how much time does it take for the new miner to get the response when the network is not under influence of any attack. The results are summarized in the following table.

TABLE VI- Mean Response times for different attack cases
[values reported on a 95% confidence interval]

Case		**Mean No. of attack packets sent**	**Mean Response time of the victim [seconds]**	
			When miner under attack selected to send response to the blockchain request	**When another miner selected to send response to the blockchain request**
No attack		-	-	5
SYN flood with IP spoofing	Spoofed IP reachable	612195	24.2 ± 4.41	5
	Spoofed IP unreachable	1643336	32 ± 3.46	5
SYN Flood with Randomized IP Spoofing		926612	25.4 ± 4.67	5
LAND Attack		1567141	30.9 ± 2.99	5

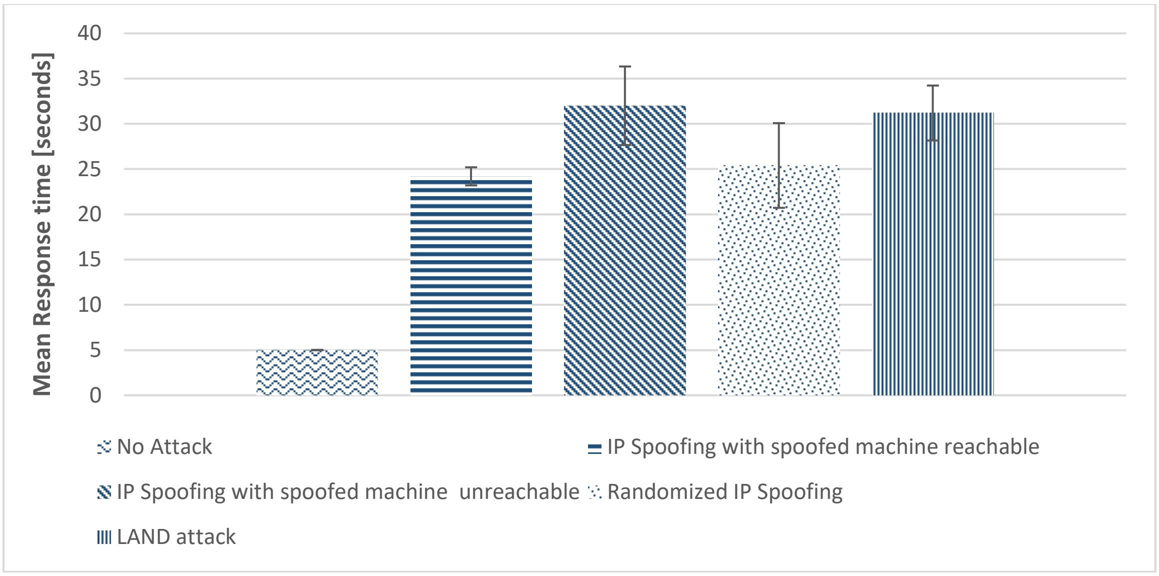


Figure 3.11. Graphical representation of mean response times in case of no attack and different subcases of the SYN flood attack with 95% confidence interval

The time-series graphs for the case with no attack and the case with maximum impact on the response time (SYN flood with an unreachable Spoofed IP) are provided in the following figures.

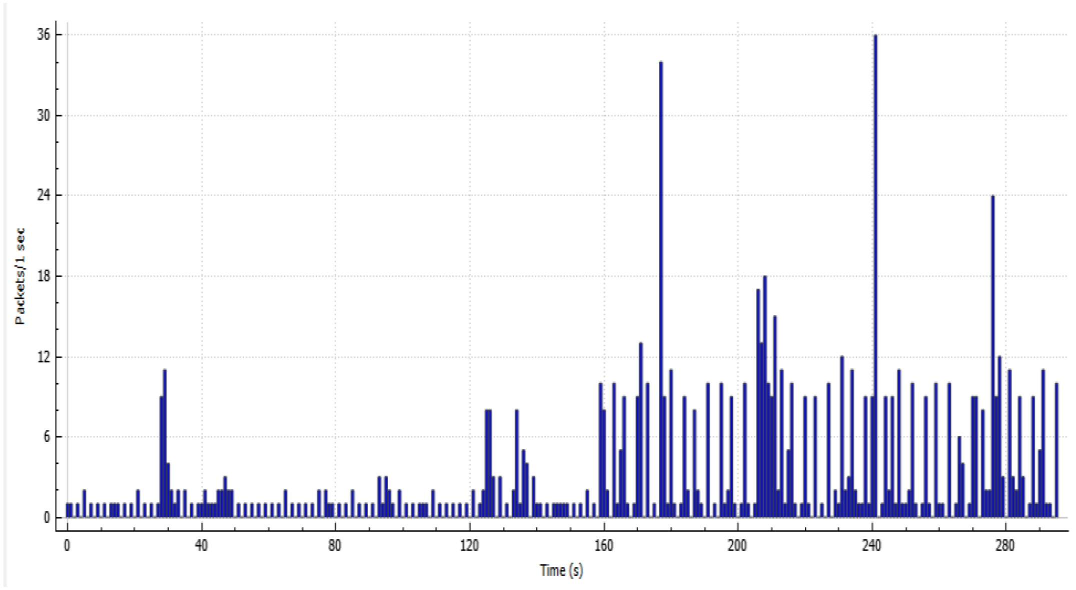


Figure 3.12. Time-series graph for Scrybe mining rounds with no attack

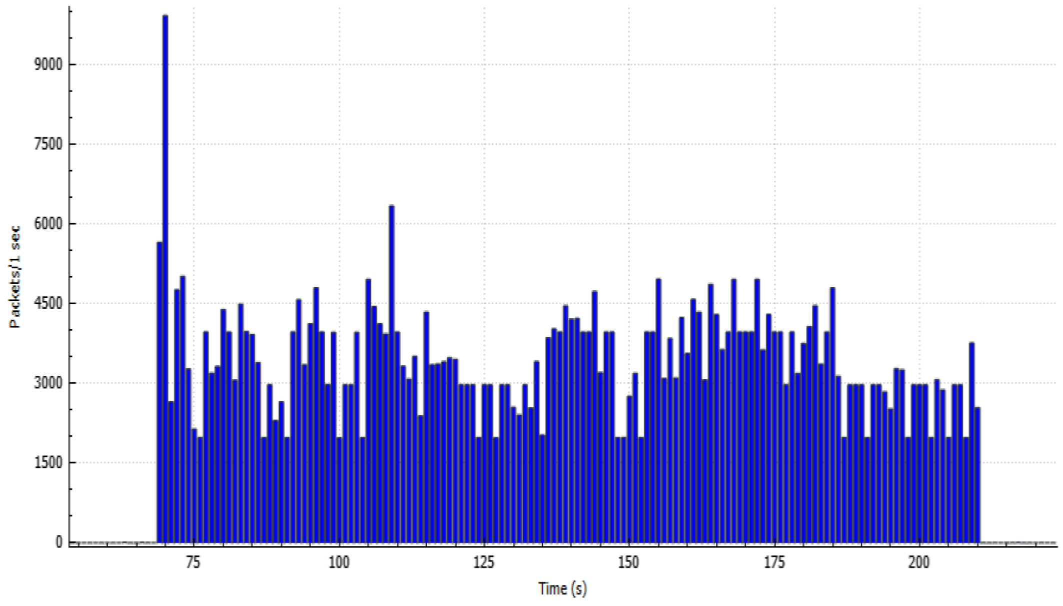


Figure 3.13. Time-series graph for Scrybe mining rounds with SYN Flood attack with unreachable spoofed IP address

Observations:

From Table IV and Figure 3.11, we observe that the network performance statistics change under the influence of an attack by a considerable extent. The attack with maximum impact is the SYN flood using IP spoofing with Spoofed machine not reachable.

3.4 Effect Of Increasing The Size Of The Attack Packets On The Impact Of DoS Attacks

We also study the effect of increasing the size of the attack packets on the impact of the DoS attacks.

For this, we consider the attack case which resulted in the maximum impact, i.e. SYN flooding using Spoofed IP with spoofed machine un-reachable. The size of attack packets is determined using the *d* parameter in the hping3 command.

The command used is: **hping3 -V -p 10987 -S -d X –flood 192.168.10.150 -a 192.168.10.134**

where *X* is the size of the packets. We tested the system for X = 120, 300, 600,900 and 1200 and determined the response time when a new miner requests for a blockchain update.

The results of these tests are summarized in Table VIII.

Table VII: Effect of increasing size of attack packets
[Values reported with 95% confidence interval]

Size of the attack packets[bytes]	**Mean Response Time of the victim [seconds]**	
	When miner under attack selected to send response to the blockchain request	**When another miner selected to send the response to the blockchain request**
120	32 ± 3.46	5
300	33.4 ± 2.63	5
600	35 ± 1.93	5
900	38.44 ± 3.33	5
1200	43.33 ± 4.68	5

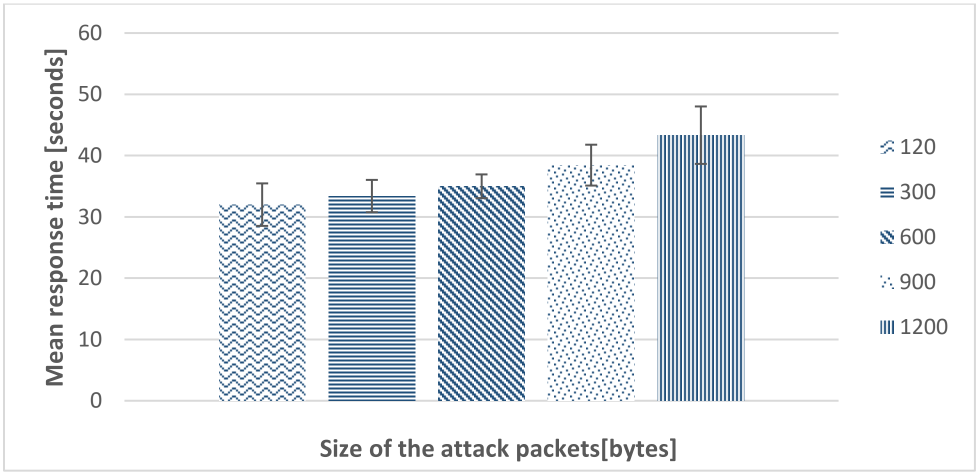


Figure 3.14. Graphical representation of effect of size of attack packets on the impact of the SYN flood attacks

CHAPTER 4

Analysis

From results in **Sections 3.3 and 3.4**, we observe that the network performance statistics change under the influence of an attack by a considerable extent. When the miner which gets selected to send the response to the blockchain request is under the influence of an attack, the response time increases. The reason for the increase of response time is the incomplete SYN connections in the victim's TCP queue as a result of a large number of SYN packets sent by the attacker. On receiving the SYN packets, the victim responds with the SYN-ACK packet for each of those; and keeps waiting for the final ACK packet from the node which initiated the SYN request. Until the TCP queue is reaped from the incomplete connection entries, the victim's resources get throttled causing it to refuse new connection requests. As a result, it is not able to process the request of a legitimate user while under the influence of an attack.

The attack with maximum impact is the SYN flood using IP spoofing with Spoofed machine unreachable. Maximum number of the attack packets are sent in this case to the victim until it again becomes responsive. The reason lies in the TCP parameter 'syn-ack-retries'. The victim is configured to keep retransmitting the SYN-ACK packet in response to the initial SYN request for a certain number of times as configured on the kernel. Since the attacker used a spoofed IP address, and kept that machine powered off, the victim has to spend an additional time in trying to reach the machine first. In addition to not sending the ACK packet, the machine does not even send an RST packet like in the case of reachable spoofed IP, since it is powered off. The server keeps retrying to reach the spoofed IP by sending the SYN_ACK packet a number of times as configured in the tcp_syn_ack_retries parameter[30]. In all of the attack cases with victim being the selected miner to send the blockchain update response, the performance of the network is degraded.

The effect of the size of the attack packets is summarized in Table VII and Figure 3.14. As the size of the attack packets is increased, the attacks become more effective causing the response time to increase. At a packet size of 1200 bytes, the response time becomes equal to that of the worst-case scenario we achieved in Table VI, i.e. the SYN flood using IP spoofing with Spoofed machine not-reachable. It should be noted that in each of the cases of packet-size, we have used a value less than the Maximum Transmission Unit [MTU]. For our network [Ethernet], the MTU is 1500.

This raises an important question about the reliability of distributed systems. Even though blockchains claim to ensure security and immutability, delay in the response of the network can cause many harmful implications. The stakes are higher for critical applications such as e-banking, healthcare and other important infrastructure.

However, an important observation is that when the miner which gets selected to send the response to the blockchain request is not the victim of the DoS attack, the response time is same as when no attack is launched. This means potentially, the attacker failed to launch a DoS attack on our network in this case. This implies that the probability of users experiencing denial-of-service while using our network is equal to the probability that the selected miner to send blockchain update responses is the victim.

Mathematically,

P(successful DoS attack on Scrybe) = P(selected miner to send blockchain updates to be the victim)

Since miners are randomly selected, the probability of selection is uniformly distributed.

$$=> \quad P(\textit{selected miner to send blockchain updates to be the victim}) = \frac{1}{N} = \frac{1}{5}$$

Therefore, for our experiments

$$P(\textit{successful DoS attack on Scrybe}) = \frac{1}{5}$$

As *N* increases, the probability of a successful DoS attack on Scrybe decreases for a constant number of miners under attack simultaneously.

4.1 Role Of Syn-Cookies

An important point to lay emphasis on is the SYN flood attacks were successful despite the SYN cookies being enabled. SYN cookies are generally considered useful against SYN flood attacks because of their stateless nature. These are configured in the ***/etc/sysctl.conf*** file and take integer values as arguments. A value of 1 means these are enabled and would be active in case a SYN flood occurs and help the server to avoid dropping connections when the SYN queue gets filled up. These are enabled by adding the following in the ***/etc/sysctl.conf*** file:

net.ipv4.tcp_syncookies = 1

Conventionally, when a client sends a SYN packet, an entry is made in the TCP queue for that regardless of the final ACK packet transmission. Enabling SYN cookies involves storing the initial SYN entry in an encoded form into the sequence number sent in the SYN-ACK packet. If the initial host responds back with a final ACK packet to complete the connection, it would use the sequence number equal to the sequence number used in the SYN-ACK packet incremented by 1.Upon receiving the ACK packet, the server decodes the sequence number and is able to reconstruct the SYN queue entry to proceed with the connection as in a conventional TCP protocol.

However, there are two major caveats to enabling SYN cookies. Server has a limit of only 8 unique MSS values since SYN cookies use 3 bits to encode that information in the SYN queue entry. In addition, the server must reject all TCP options (such as large windows), because the SYN queue entry where this information is otherwise stored is discarded by the server.

Another important implication of using SYN cookies is an additional computational overhead. Implementing SYN cookies involves a cryptographic hash of the IP address and the port number used on the server and the timestamp while sending the responses to the SYN requests.

This results in an extensive overhead. When attacks occur and the SYN cookies are enabled, the server has to spend a lot of efforts encrypting responses to the incoming SYN requests. This also results in a slower response by the server to the requests from legitimate users.

SYN cookies are not capable of reducing or deflecting traffic. Therefore, SYN cookies are not necessarily useful against attacks targeting bandwidth. The attacks launched on Scrybe involved sending thousands of SYN packets per second, similar to a bandwidth attack.
We'll try to optimize the utility of enabling SYN cookies to mitigate the impact of SYN floods using the ***tcp_syn_ack_retries*** and the ***tcp_max_syn_backlog*** kernel parameters in the next chapter.

4.2 Optimization Of Kernel Parameters To Strengthen Defense Of Syn Cookies Against DDoS Attacks

We tried to study the effect of changing the ***tcp_syn_ack_retries*** and the ***tcp_max_syn_backlog*** parameters to help SYN cookies offer better protection against the SYN flood attacks.Both of these kernel parameters are tunable. We choose a These parameters are also configured in the ***/etc/sysctl.conf*** file.

- **tcp_syn_ack_retries :**

This parameter determines the number of times the server should retry to send the SYN-ACK packet in response before giving up waiting for the final ACK packet. On Linux machines, the default is 5. We tested the system performance by lowering this value and checking the response time while under the influence of D/DoS attack. The results are for the worst-case scenario we achieved in Table VI, i.e. the SYN flood using IP spoofing with Spoofed machine not-reachable.
Table VIII provides the summary of the results.

Table VIII: Effect of decreasing tcp_syn_ack_retries value
[Values reported at 95% confidence interval]

tcp_syn_ack_retries value	**Response time[seconds]**
5	32 ± 3.46
4	29.7 ± 3.62
3	28.33 ± 3.42
2	26.22 ± 3.66
1	22.55 ± 3.48
0	15.44 ± 2.05

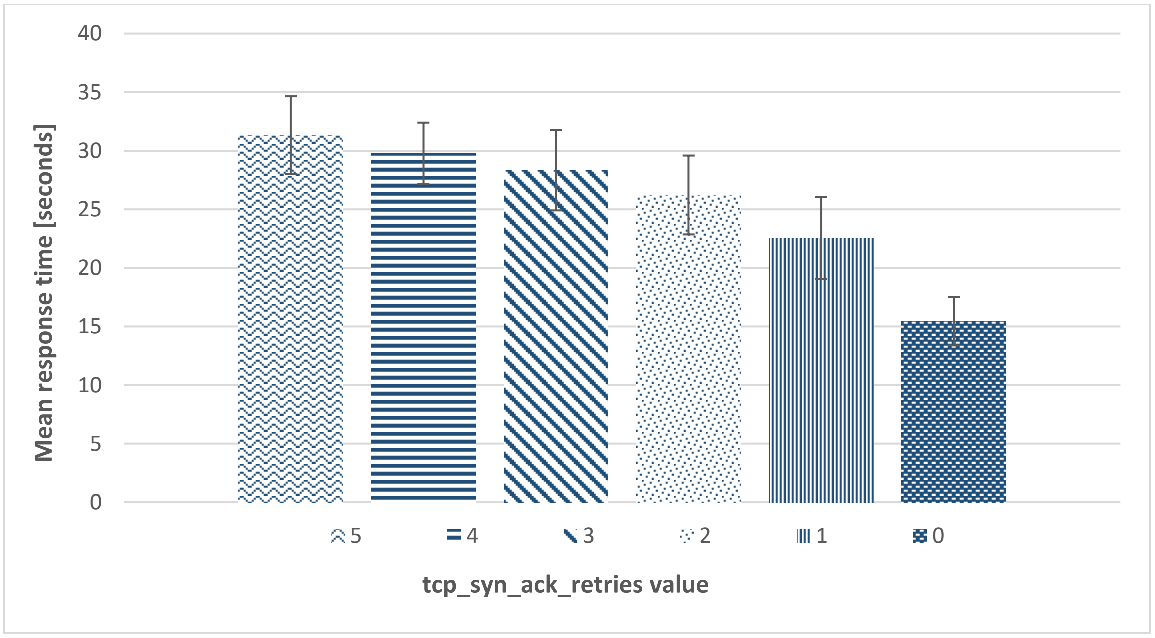


Figure 4.1. Graphical representation of the effect of decreasing the tcp_syn_ack_retries parameter value

Observations:

The performance of the system increases while being attacked if the number of syn_ack_retries value is reduced. In other words, the server doesn't wait for a longer time before terminating the connection request. The retry times are calculated based on the Round Trip Time [RTT] between the peers. For linux systems, it is around 2 seconds for the first packet, doubled to 4 seconds after second retry packet sent, then 8 seconds for the next packet, and so on [31]. Configuring the server to a zero number of syn_ack_retries improves the performance considerably during the SYN flood attacks. However, there is a trade-off. The connections with legitimate nodes can suffer network congestion or link failure momentarily and the SYN-ACK packet might not reach them. The server should be able to attempt retransmitting the packet if need be. For that reason, we set this value to an optimum value of 3.

Next, using the syn_ack_retry value of 3, we try to optimize the system performance changing the tcp_max_syn_backlog parameter as follows:

- **tcp_max_syn_backlog:**

This parameter determines when to activate the syn cookies. When the TCP connection requests exceed the value set in this parameter, the system starts using the syn cookies. As discussed in Section 5.1.1, enabling syn cookies (setting a value of 1) means those are enabled and can be activated as required. On Linux machines, the default value is 128 (2^7).

We studied the effect of increasing the value of this parameter.

The results are summarized below:

Table IV: Effect of increasing tcp_max_syn_backlog value
[values reported at 95% confidence interval]

	tcp_max_syn_backlog value	**Mean response time [seconds]**
tcp_syn_ack_retries =3	127	28.33 ± 3.42
	256	23.77± 4.0
	512	18.55 ± 2.74
	1024	17.33 ± 1.59
	2048	11.55 ± 1.29

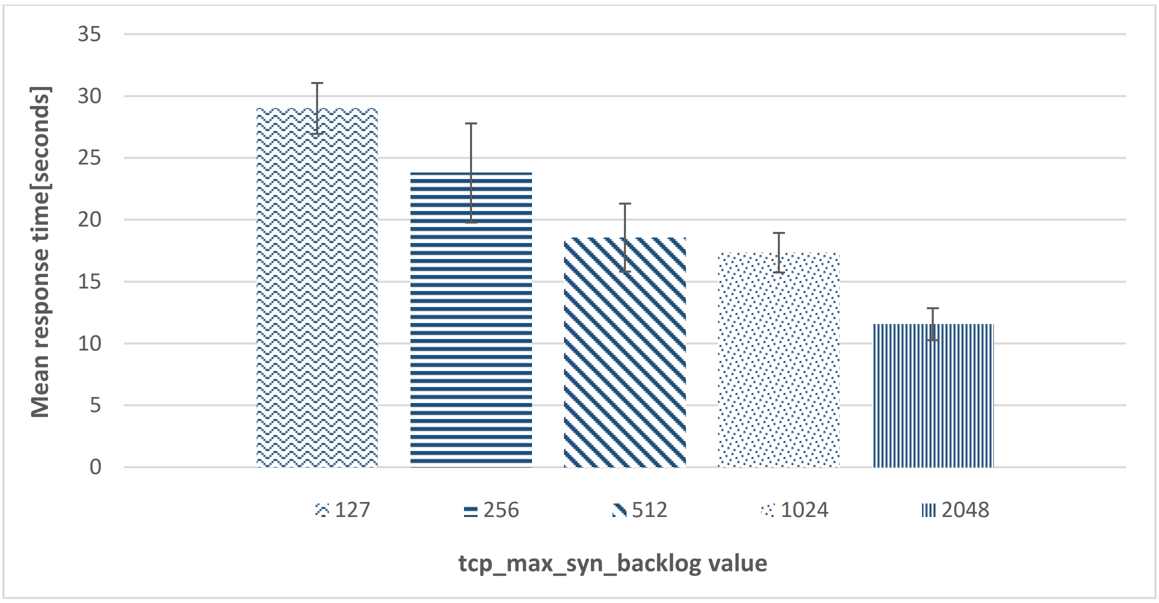


Figure 4.2. Graphical representation of the effect of decreasing value of the tcp_syn_backlog

Observations:

The performance of the system improves with the increasing number of the tcp_max_syn_backlog value. This is because an increased value must mean the TCP queue saturates at a higher number of flood packets in case of a SYN flood attack, thereby, responding faster to the legitimate connections. In addition to this, a higher value means that the syn cookies are not activated until the TCP connection requests exceed 2048. As discussed earlier, syn cookies increase computational overhead and slow the performance during the flooding attacks collaterally.

We observe that the response time goes down to 11 seconds for the tcp_max_syn_backlog value of 2048 as compared to 31 seconds on the default value of 127.

Like in the case of tcp_syn_ack_retries parameter, arbitrarily increasing the value of this parameter is not a reasonable idea. Each entry in SYN queue takes 256 bytes and in case a SYN flood is launched, all those resources would be wasted on storing the attack packets. Therefore, it is not a good idea to configure the max_syn_backlog to a very large value.

The overall improvement as a result of optimizing both the tcp_max_syn_backlog value and the max_syn_ack_retries value can be seen by comparing the mean response time of about 32 seconds before optimization to about 11 seconds after optimization.

CHAPTER 5

Lightweight Mining Consensus

As described in Chapter 1, Malicious miners and arbitrary faults might disrupt the functioning of the blockchain network using multiple strategies. They might share conflicting hash values or secret values to different miners causing confusion and try to favor a certain outcome of the modulo-n addition operation. They might also exclude certain client files, or files from certain clients, thereby, denying them the service they have employed the blockchain technology for However, as long as there is at least a single honest miner in the network, faults can't disrupt the network. In a practical blockchain network, there would always be more than one honest miner. In this work, we showed a single honest miner is enough to keep a check on faulty activities in a blockchain network. As number of honest miners increases, so does their control of the consensus in a blockchain network.

In this chapter, we verify that as long as there is at least one host miner in the network, consensus is reached, and the client transactions are registered on the blockchain. The detailed algorithm is provided first. Then we prove theorems and discuss cases which prove our hypothesis that a single honest miner is sufficient to keep a check on the faulty activities in a blockchain network.

Lightweight Mining (LWM)

Lightweight Mining Algorithm (LWM)
Input: The number of miners N.
Algorithm: For each miner m_i, $0 \le i \le N-1$,

- *Step 1*: m_i generates a secret value (random number) s_i ;
- *Step 2*: m_i broadcasts the SHA-3 hash of the s_i, denoted by $H(s_i)$ after signing it with its private key P_i.
- *Step 3(a)*: Wait for $\frac{n}{3}$ hashes and proceed to Step 3(b).
- *Step 3(b):* Broadcast the $\frac{n}{3}$ hashes received in Step 3(a) to all the miners. Once values received from 50% + 1 number of miners, check that no conflicting values have been received corresponding to a particular miner. If yes, drop those values and evict those miner(s) from the current round. Once verification done, proceed.
- *Step 4*: m_i broadcasts the secret value s_i signed with its private key P_i.
- *Step 5(a)*: Wait for $\frac{n}{3}$ random numbers and proceed to Step 5(b).
- Step 5*(b):* Broadcast the $\frac{n}{3}$ secret values received in Step 5(a) to all the miners. Once values received from 50% + 1 number of miners, check the following:
    - No conflicting values have been received corresponding to a particular miner.
    - Verify that the secret value corresponds to the hash shared in Step 3(a).

  If discrepancies found in any of these two checks, drop those values and evict the miner(s) involved from the current round.
  Once check completed, proceed to the modulo-addition operation.
- Step 6: m_i calculates $l = \sum_{j=0}^{N-1} s_j \bmod N$
  C is the consensus on the selected miner, l, to create the new block from the collected client files. (Without loss of generality, we map $m_i = j$, $0 \le j \le N-1$ as a simple rank ordering for the registered miners.)

5.1 Overview

In this paper, we focus on verifying the robustness of our blockchain i.e. the ability of nodes to converge to a decision despite the presence of arbitrary errors. This is most easily modeled, by allowing the presence of malicious nodes which we could try to disrupt the functioning of the algorithm. Lamport's work explains that other failure models are overly optimistic[28].

To guarantee that *Scrybe* reaches consensus, we rely on Lamport's Byzantine Fault Tolerance (BFT) results. BFT work proves [38] that it is impossible to reach consensus if the number of faulty nodes, *f,* in a distributed network is not less than $\frac{1}{3}$rd of the total nodes, *n*. Though BFT handles the final outcome to be either '0' or '1', we and we try to reach a consensus on a value 0 to n-1, where we need to show both consensus and that the value is an emergent property of the set of nodes such that no group of nodes can influence the choice as long as any single node does not collude with that group. BFT needs $\frac{1}{3}$rd of *n to* have the same value, *S*. We need responses from $\frac{1}{3}$rd of *n*, regardless of the values.

We accept Lamport's result that the number of malicious nodes must be less than $\frac{1}{3} * n$. This may not be the largest number of malicious nodes we can tolerate, that proof is outside the scope of this paper. Using a larger number would slow down our algorithm. Using a smaller number may be unrealistic. We accept the $\frac{n}{3}$ value as one that is widely used in this domain and practical for our system. Given Lamport's result, we limit the size of the colluding set to less than $\frac{1}{3}$rd of the participating nodes, and therefore waiting for $\frac{1}{3}$rd of the *n* possible values ensures that at least one value is from an honest miner.

5.2 Reaching Consensus In Scrybe

In *Scrybe's* lightweight mining process, there are three main phases where information is shared: the hash-sharing phase, the secret-sharing phase, and the block-sharing phase:

1. **Hash-sharing phase:** Each miner, m_i, generates a random number, called a secret-value, $s_{i,.,}$ computes its hash $H(s_i)$, signs $H(s_i)$ using its private key , P_i , and shares the signed hash with all other participating miners.

    a. Once $\frac{1}{3}$rd of the *n* number of hashes is received on a node, we proceed to Step b.

    b. In this step, all the miners broadcast all the $\frac{n}{3}$ hashes they received in Step a. This ensures that all the miners agree on a set of values.

Once values from 50 % + 1 number of the miners are received, thereby, ensuring the network has not fractured, we verify the following:

- No conflicting values have been received corresponding to a particular miner. If yes, drop those values, identifying the miner(S) with their private key(s), and evict them from the current round.

Once verification done, proceed to Step 2.

2. **Secret-sharing phase:** In this phase, each miner is expected to share its secret value *s_i* , signed with its private key,*P_i*, with all the miners.

   a. Once $\frac{1}{3}$rd of *n* secret values are received, we proceed to Step b.
   b. In this step, all the miners broadcast the $\frac{n}{3}$ secret values they received in Step a.

This ensures that all the miners have the common majority of values required to run the modulo-n operation. Once values from 50 % + 1 of the miners are received, thereby, ensuring that the network has not fractured, we check the following:

- No conflicting values have been received corresponding to a particular miner.
- Verify that the secret value corresponds to the hash shared in Step 3(a).

Once check is completed, the modulo-n operation is implemented on each node to select the miner to create the new block.

3. **Block-sharing phase:** Once consensus is reached on which miner would create the new block *B_i*, the selected miner broadcasts the new block, signed with its private key with all the miners who participated in Step 2. After all the miners verify the validity of the block, they append it to their own copies of the blockchain.

5.3 Cases

I. **An honest miner gets selected to create the new block:** This is an ideal situation as an honest miner would ensure all the client transactions since last block are included in the current block. Thus, clients are provided the services they have employed the blockchain for. After creating the block, the honest miner signs in and broadcast it to the rest of the miners who verify it and add it to their own copy of blockchain.

II. **A malicious miner gets selected to create the new block:** The modulo-n addition can result in a malicious miner to get selected to create the new block. A malicious miner can choose to exclude transactions submitted from a specific client or clients, thereby denying them the services they are authorized for. However, when the new block is created, the selected miner needs to sign it with its key and broadcast it. If any transaction has been left out, the honest miner(s) would be able to detect it upon verification even if the other malicious miners form a coalition with the selected malicious miner. Hence, due to failing to prove the validity, the block would be deleted. Since the invalid block was signed by the malicious miner, we know who the miner was and evict them from the current mining round.

Theorem : Given a set of *n* nodes, each with a secret value *S(j)*, where at most $\frac{n}{3}$ are faulty, the non-faulty nodes can reach consensus C on a value *l*, where *l* is the miner selected to create the new block, and = $\sum_{j=0}^{N-1} s_j \, mod \, N$.

Proof: We begin by proving that as long as there is at least one honest miner in the network, the selection of the miner to create the next block is random and can't be controlled by the malicious miners. To prove this, we show if values from n -1 miners result in a particular output of the modulo-n operation, an honest miner can choose the secret value in a way which forces the summation to whatever we want. In practice, the miners bind to a value and the sum can't be forced to a desired value at any cost. But this proof describes that having at least a single value from an honest miner randomizes the output and makes it fair.

Without loss of generality, let us assign the honest miner m_n an index 0. Hence, the non-malicious nodes are in the range $0 < j < N$. All the nodes N_j share hash values $H(s_i)$ and secret values s_j for $0 < j < N$. Let us assign $l = \sum_{j=0}^{N-2} s_j \, mod(N\text{-}1)$ as the modulo-n operation result of the values chosen by the malicious miners. If an honest miner picks an arbitrary integer value k from the range 0 to $N\text{-}1$, we can assign the secret value for the honest miner as $s_0 = (l - k) mod\ N$ that forces the summation in the modulo-n operation to be whatever we want.

Thus, the result is not controlled by the malicious miners anymore because the honest miner would have to agree with the values generated on the malicious nodes.

Axiom: As long as there is at least a single honest miner in the network generating a random number from range 0 – n-1 with uniform distribution, all the results are equally likely.

Theorem: We can always detect if a malicious miner sends conflicting values to different miners

Proof: The value can be received and promulgated by either a single node or multiple nodes. Step 1.b of the LWM involves broadcasting the $\frac{1}{3} * n$ hashes received in Step 3.a to all the miners. If only one node gets the value, then all nodes get that value, so there is no conflict. Another scenario could be that multiple miners receive the same value which also results in no conflict. However, if conflicting values are sent out to different miners, the broadcast step allows nodes to drop those values.

Once values from 50% + 1 miners are received, we proceed to next phase. This means, the hashes sent out by the malicious miners would be received at each node from multiple sources. Comparing the hashes corresponding to each miner, an honest miner would confirm if conflicting values have been sent out. Since all the hashes need to be signed, we can determine the miner(s) who sent out conflicting values and evict them from the network for the current round. The algorithm proceeds with the remaining validated values.

Similarly, a malicious miner can't go undetected if they send conflicting random numbers (secret values) to different miners.

Mathematically, if a miner m_i sends a hash $H(s_i)$ to one miner, the hash value corresponding to m_i as $H(s_i)$ after the broadcast step of 3.b. should be same for all the miners.

$$H(s_i) = H(s_j) \text{ iff } i = j$$

As long as there is at least one honest miner, this condition is satisfied.

Theorem: We can always detect if a malicious miner sends a secret value which doesn't correspond to the hash shared in the hash-sharing phase

Proof: Step 2.a requires miners to send the hashes first before sending the actual secret value. Both the hashes and secret values are signed by the miner's private key. If a malicious miner sends a secret value which does not correspond to the hash they sent previously, an honest miner would confirm this activity upon comparison of the hash generated by the secret value shared by the miner with the hash the miner shared themselves previously. The signature would determine which miner did it and they are evicted from the network for the current round. The algorithm proceeds with the validated values.

Mathematically, if a miner m_i sends out a secret number s_i in the secret-sharing phase, hash of the number, i.e. $H(s_i)$ should be equal to the hash the miner m_i shared in the hash-sharing phase. This is guaranteed as long as there is at least one honest miner in the network.

CHAPTER 6

Conclusion

This thesis focuses on the security analysis of a blockchain network, *Scrybe*, evaluating its robustness and availability. To do this, common approaches are considered for disabling, or degrading distributed systems. The most successful tools are generally referred to as Distributed Denial of Service (DDoS) attacks. To verify our proposed blockchain network, we therefore integrate successful DDoS methods into our verification suite. Since *Scrybe* uses TCP for communication, we leverage DoS attacks as a tool to exploit the known flaws in the TCP such as the SYN flood D/DoS attack.

We observe that the presence of D/DoS attacks increases the response time of the network. We also observe how the performance changes with change in some Linux kernel TCP parameters. Consensus is reached in all the cases.

We also consider the possibility of a malicious miner(s) attempting to disrupt the functioning of the blockchain network so that client files are not successfully registered on the network. We verified that as long as there is at least a single honest miner in the network, consensus on who gets to create the new block is guaranteed and the client files are registered on the network.

6.1 Future Work

The experiments and results are valid for our test case scenario of one malicious miner out of five malicious miners. As the total number of participating miners increases, and more malicious miners are present, the impact of the attacks could be more powerful because they could be launched by proper coordination of the malicious miners. They could attack multiple miners simultaneously, thereby, increasing the probability of the miner getting selected to send the blockchain update response to be under the influence of the attack. As a result, the probability of legitimate miners or clients experiencing denial-of-service can be increased.

We verified how consensus is reached as long as there is at least one honest miner in the network. In practice we would have an *n* number of miners with less than $\frac{1}{3} * n$ faulty. We need to verify the minimum number of *n* for which our tests and proofs are valid.

Bibliography

[1] Naazira et. al, “Evaluation and Design of Performable Distributed systems,” *Handbook of Performability Engineering, Springer* , 2020 [*Accepted*].

[2] Worley C. et al., “Scrybe: A Second-Generation Blockchain Technology with Lightweight Mining for Secure Provenance and Related Applications”, In: Choo KK., Dehghantanha A., Parizi R. (eds) *Blockchain Cybersecurity, Trust and Privacy. Advances in Information Security, vol 79. Springer,* 2020.

[3] S. Nakamoto, “Bitcoin: a peer-to-peer electronic cash system”**,**2008.

[4] S. Medury, A. Skjellum, R. R. Brooks, Lu Yu, “SCRaaPS: X.509 Certificate Revocation using the Blockchain-Based Scrybe Secure Provenance System,”*Malware 2018,* Nantucket, October 2018.

[5] Jonathan Oakley, Carl Worley, Lu Yu, Richard Brooks, Anthony Skjellum, “Unmasking Criminal Enterprises: An Analysis of Bitcoin Transactions,” *Malware 2018,* Nantucket,October 2018.

[6] O. Hambolu, L. Yu, J. Oakley, R. R. Brooks, U. Mukhopadhyay, and A. Skjellum, “Provenance threat modeling,” In Privacy, Security and Trust (PST), *IEEE 14th Annual International Conference on Privacy, Security and Trust,* 2016.

[7] U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu, and R. R. Brooks, “A brief survey of cryptocurrency systems,” In Privacy, Security and Trust (PST), *IEEE 14th Annual International Conference on Privacy, Security and Trust,* November 2016.

[8] P.J. Taylor et al., “A systematic literature review of blockchain cyber security”, *Digital Communications and Networks,Elsevier,* 2019.

[9] M. Samaniego, R. Deters, “Blockchain as a service for IoT”, I*EEE International Conference on Internet of Things (iThings)* and *IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)* and *IEEE Smart Data (SmartData)* p. 433-436, 2016

[10] L. Kan, Y. Wei, A. Hafiz Muhammad, W. Siyuan, G. Linchao, H. Kai,“A multiple blockchains architecture on inter-blockchain communication”**,** *IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*,pp. 139-145, 2018.

[11] https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/

[12] https://searchnetworking.techtarget.com/definition/TCP

[13] http://www.firewall.cx/networking-topics/protocols/tcp/138-tcp-options.html

[14] https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/

[15] https://ddos-guard.net/en/terminology/protocols/tcp-3-way-handshake

[16] https://www.guru99.com/tcp-3-way-handshake.html add RFC reference

[17] Debra Littlejohn Shinder Michael Cross, “Scene of the Cybercrime”,2nd Edition, Elsevier, June, 2008.

[18] Ph.D., Louisiana State University, Dissertation: Reliable Sensor Fusion Algorithms: Calibration and Cost Minimization, Adviser: S. S. Iyengar, 1996, Computer Science.

[19] R. R. Brooks and S. S. Iyengar, Multi-Sensor Fusion: Fundamentals and Applications with Software, (1998) Prentice Hall PTR, Upper Saddle River, NJ.

[20] R. R. Brooks and S. S. Iyengar, "Robust Distributed Computing and Sensing Algorithm", *IEEE Computer,* 29 (6), 53-60 (1996).

[21] B. Ao, Y. Wang, L. Yu, R. R. Brooks and S. S. Iyengar, “On Precision Bound of Distributed Fault-Tolerant Sensor Fusion Algorithms,” *ACM Computing Surveys*, 49(1), 5:1- 5:23, (2016).

[22] L. Liang, K. Zheng, Q. Sheng and X. Huang, "A Denial of Service Attack Method for an IoT System," *8th IEEE International Conference on Information Technology in Medicine and Education ITME),* Fuzhou, pp. 360-364, 2016.

[23] Kshirsagar, D., Rathod, A., & Wathore, S., “Performance analysis of DoS LAND attack detection”, *Perspectives in Science*, 8, 4–6. https://doi.org/10.1016/j.pisc.2016.06.074 , 2016.

[24] M. Vasek, M. Thornton, and T. Moore, “Empirical analysis of denial-of-service attacks in the bitcoin ecosystem,” in *International Conference on Financial Cryptography and Data Security, Springer*, pp. 57–71, 2014.

[25] “Distributed Denial Of Service-Defense Attack Tradeoff Analysis (DDoS -Data)”, *Johns Hopkins University – APL , Sponsored by Defense Advanced Research Projects Agency DARPA*, Order No. M101, December 2004.

[26] Ye, G. Li, H. Cai, Y. Gu and A. Fukuda, “Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting”, *IEEE 5th International Conference on Dependable Systems and Their Applications,* Dalian, pp. 15-24, 2018.

[26] K. Geetha and N. Sreenath, “SYN flooding attack — Identification and analysis”, IEEE International Conference on Information Communication and Embedded Systems (ICICES2014), Chennai, pp. 1-7, 2014.

[28] S.Acharya and N.Tiwari, “Survey Of DDoS Attacks Based On TCP/IP Protocol Vulnerabilities”, *IOSR Journal of Computer Engineering (IOSR-JCE)* e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 18, Issue 3, Ver. IV, pp. 68-76, June 2016.

[29] M. S. Al-Hawawreh, "SYN flood attack detection in cloud environment based on TCP/IP header statistical features," IEEE *2017 8th International Conference on Information Technology (ICIT)*, Amman, pp. 236-243, 2017.

[30] D.Kshirsagar, S.Sawant , A.Rathod and S.Wathoee, “CPU Load Analysis & Minimization for TCP SYN Flood Detection”, *Procedia Computer Science*, Volume 85, pp. 626-633, 2016.

[31] Andrew S. Tanenbaum , “Computer Networks”, *Pearson Education*, 2012.

[32] T. Jamal, Z. Haider, "Denial of Service Attack in Cooperative Networks", *Proc. of ArXiv arXiv: CoRR* [computing research reposititory on arXiv], Oct. 2018.

[33] D.Nashat , X. Jiang, S. Horiguichi, “Detecting SYN Flooding agents under any type of IP Spoofing”, *IEEE International Conference on e-Business Engineering*, pp. 499-505, 2008.

[34] W. Stevens, “TCP/IP Illustrated”, *Pearson Education*, 2016.

[35] https://blog.cloudflare.com/syn-packet-handling-in-the-wild/

[36] Leslie Lamport, R.Shostak,M.Pease, “Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems*,Vol. 4, No. 3, July 1982, pp. 382-401