NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

COMPARISON RESEARCH FOR OS ETHEREUM BLOCKCHAIN AND IBM ENTERPRISE-LEVEL HYPERLEDGER TECHNOLOGY APPLIED IN AUTONOMOUS NAVY UNCLASSIFIED SOFTWARE DISTRIBUTION BASED ON BLOCK TIME AND SCALABILITY

by

Nur Endah Dwijayanto

September 2020

Thesis Advisor: James B. Michael
Co-Advisor: Peter R. Ateshian

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
COMPARISON RESEARCH FOR OS ETHEREUM BLOCKCHAIN AND IBM ENTERPRISE-LEVEL HYPERLEDGER TECHNOLOGY APPLIED IN AUTONOMOUS NAVY UNCLASSIFIED SOFTWARE DISTRIBUTION BASED ON BLOCK TIME AND SCALABILITY

5. FUNDING NUMBERS

6. AUTHOR(S) Nur Endah Dwijayanto

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

The U.S. Navy seeks to leverage emerging technologies to manage massive amounts of data from multiple geographically separated systems. It is aware of the importance of data usage and data transfer in supporting its operations. Data management requires a data transfer system that is safe, fast, and scalable. Autonomous Navy Unclassified Software Distribution (ANUSD) is an application for delivering software to all nodes on the Navy's enterprise network based on blockchain technology. Blockchain is the right candidate and emerging solution to ensure the triad of confidentiality, integrity, and availability. In this thesis, we perform a comparison of public blockchain and private blockchain with the aim of determining which one would perform better in conjunction with ANUSD. We used an IBM Hyperledger (private blockchain) network and an Ethereum blockchain (public blockchain) network as the basis of the comparative analysis of their latency and scalability. We compared the transactions per second (TPS) achieved with Ethereum against that of Hyperledger with the ANUSD application installed. The results showed that as we scaled up the Ethereum network, there was a significant increase in TPS. In contrast, increasing scalability did not have a significant impact on TPS for the Hyperledger network.

14. SUBJECT TERMS blockchain, IBM Hyperledger, Ethereum blockchain	**15. NUMBER OF PAGES** 95
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

COMPARISON RESEARCH FOR OS ETHEREUM BLOCKCHAIN AND IBM ENTERPRISE-LEVEL HYPERLEDGER TECHNOLOGY APPLIED IN AUTONOMOUS NAVY UNCLASSIFIED SOFTWARE DISTRIBUTION BASED ON BLOCK TIME AND SCALABILITY

Nur Endah Dwijayanto
Lieutenant Commander, Indonesian Navy
EE, Indonesian Naval Academy, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2020

Approved by: James B. Michael
Advisor

Peter R. Ateshian
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The U.S. Navy seeks to leverage emerging technologies to manage massive amounts of data from multiple geographically separated systems. It is aware of the importance of data usage and data transfer in supporting its operations. Data management requires a data transfer system that is safe, fast, and scalable. Autonomous Navy Unclassified Software Distribution (ANUSD) is an application for delivering software to all nodes on the Navy's enterprise network based on blockchain technology. Blockchain is the right candidate and emerging solution to ensure the triad of confidentiality, integrity, and availability. In this thesis, we perform a comparison of public blockchain and private blockchain with the aim of determining which one would perform better in conjunction with ANUSD. We used an IBM Hyperledger (private blockchain) network and an Ethereum blockchain (public blockchain) network as the basis of the comparative analysis of their latency and scalability. We compared the transactions per second (TPS) achieved with Ethereum against that of Hyperledger with the ANUSD application installed. The results showed that as we scaled up the Ethereum network, there was a significant increase in TPS. In contrast, increasing scalability did not have a significant impact on TPS for the Hyperledger network.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ANUSD	Autonomous Navy Unclassified Software Distribution
API	Application Programming Interface
AWS	Amazon Web Service
BYFN	Build Your First Network
CA	Certificate Authority
CLI	Command Line Interface
D-APPS	Decentralized Application
FTP	File Transfer Protocol
IPFS	InterPlanetary File System
Org	Organization
OS	Operating System
PBFT	Practical Byzantine Fault Tolerance
PoS	Proof of Stake
PoW	Proof of Work
REST	Representational State Transfer
SC	Smart Contract
SDK	Software Development Kit
SSH	Secure Shell
TPS	Transaction per Second
VM	Virtual Machine
VPC	Virtual Private Cloud

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Emerging blockchain technologies offer security solutions to support key aspects of the Cyber Security Triad, namely confidentiality and integrity. As explained by . Manik [1] in a recently published paper, "the basis of Blockchain technology is digital accounting of various digital transactions, in which the transaction data cannot be changed or modified even though it has spread to various network nodes."

A. PROBLEM STATEMENT

Blockchain is a permanent and unalterable transaction recording application, supported by a decentralized database on a peer-to-peer network. Among the leading platforms that have adopted blockchain technology are IBM's Ethereum, a cryptocurrency platform, and Hyperledger, an umbrella project consisting of open source blockchains, hosted by the Linux Foundation. Given the complexity of the transaction processes in such network environments, security is essential as well as challenging. Yet, as previously noted, because a blockchain has no central storage point that could be a target for attackers, security gaps are virtually impossible in blockchain technology. Also contributing to the strength of the security of this technology are consensus algorithms within blockchains, enabling all peers on a network to achieve common agreement not only about a single data value but the state of the distributed ledger and the network itself.

Based on that fact, blockchain technology seems ideally suited to the military world, especially for navies. Data security is essential in supporting their work, which involves significant amounts of confidential information. The use of blockchain technology in the U.S. Navy, for instance, could provide a breakthrough for organizational development. Securing data and transaction records is a vital measure that could be achieved with such supporting technology immediately, either through the use of the Ethereum or the Hyperledger platform. Similarly, the United States Navy should adopt a migration process from the traditional non-blockchain technology pattern of data security to blockchain technology (Ethereum or Hyperledger).

To implement this new technology in the organizational environment of the U.S. Navy, however, requires a research-based guide to provide input for leaders to choose the best technology based on organizational needs. Two essential factors that should serve as a guide in determining the best blockchain technology for the U.S. Navy are scalability and latency, and they are considered in detail in this research.

Scalability and latency are vital factors to support the United States Navy organization, which is supported by a fleet operating in a vast area. These factors become the primary concern to determine whether Hyperledger or Ethereum is the better platform.

One of the urgent needs of the U.S. Navy is a supply chain application for software transfer, enabling the distribution of new software, updates, and patches to all organization parts across the Navy environment. As Ethereum and Hyperledger are only platforms, they cannot meet this need themselves. Hence, in assessing the scalability and latency of Ethereum and Hyperledger, it is also important to consider the decentralized application that will work on each of the two platforms.

To migrate its supply chain from traditional non-blockchain technology to modern blockchain technology, the U.S. Navy must adopt Autonomous Navy Unclassified Software Distribution (ANUSD), which uses blockchain technology. It is expected to be a breakthrough that brings a more secure and modern system supply chain, so the organization's need for data security and transaction recording can be fulfilled.

This research focuses on the effects of the scalability and latency factors, also known as transactions per second, for the Blockchain network. The expected result of this research is the identification of the proper blockchain system to support the best performance in terms of transactions per second for the ANUSD application. Further, the writer hopes that this thesis can be the first step in determining the best blockchain technology for the U.S. Navy.

B. APPROACH

There are three methodologies used in this thesis. The first one is to determine the ANUSD standard architecture required by the U.S. Navy to test the transactions per second

on the two platforms (Ethereum and Hyperledger). The second method is a scalability test carried out by adding new peers to each platform from a minimum to a certain number of peers on the network. The purpose of this testing is to see the effect of adding peers to latency (transactions per second). The last method is a latency (transactions per second) test on the transaction load given with a varied file size at a certain minimum transaction value up to 10,000 transactions. It is conducted as a comparison study of the two platforms and aims to obtain accurate data to produce a conclusion that can guide Navy leaders in selecting the better blockchain platform to support ANUSD.

C. SCOPE

The thesis is limited to comparing the Ethereum Blockchain Operating System (OS) and the IBM Enterprise Level Hyperledger Framework regarding Smart Contract (SC) and Distributed Ledger Technology (DLT), based on scalability and latency (transactions per second). This research is performed on a Linux Operating System with local area network connectivity in a multi-node network. Resources used come from various digital sources and publicly available literature.

D. FINDINGS SUMMARY

After several trials using ANUSD on each network and using 10–10,000 transactions, we obtained a comparison of the latencies (transactions per second) for the two networks (Hyperledger and Ethereum). Each network gives results that show a particular pattern, as evident in the chart comparing the test results in Chapter IV. From the results, it can be concluded that increasing scalability affects the expanding value of latency (transactions per second, or TPS) on the Ethereum network. By contrast, increasing scalability does not have a significant impact on the Hyperledger network's TPS.

The evidence and rationale for this conclusion are described in Chapter IV and Chapter V. These chapters also outline several constraints, such as the limitations of the blockchain network on certain types of Amazon Web Service (AWS) and the availability of installation documentation provided by the IBM Hyperledger, which was a separate obstacle in the preparation of this thesis.

E. THESIS ORGANIZATION

Chapter II provides the necessary background for understanding blockchain technology and its aspects relevant to this thesis. Specifically, it provides an overview of Ethereum, IBM Hyperledger, the concept of decentralized applications, and Hyperledger Caliper. Chapter III discusses the design of the ANUSD application as well as the Hyperledger and Ethereum platforms to present the differences between scalability and block time on those two platforms. The implementation of the research is introduced in Chapter IV. That discussion includes the results of the trial of 10,000 transactions applied to ANUSD installed on each platform. Chapter IV provides a chart comparison of scalability and block time for the two platforms. Finally, Chapter V contains the conclusion, a summary of the findings, and suggestions for future work that can be implemented to build on this thesis.

II. BACKGROUND

In discussions of distributed block transactions that we know as a blockchain, several researchers have written hypotheses about the transaction time of the blockchain system, but no one has ever conducted research on transaction time and latency with a blockchain. Several researchers have written that blockchain technology is too slow compared to traditional systems such as client-server connectivity. In a journal article written by Yasaweerasinghelage, Staples, and Weber [2], the authors explain that "the transaction time of a blockchain is determined by many factors, including network delays, transaction costs, number of transactions, and strategic decisions made by the miners. Therefore, transaction inclusion times can vary widely." From this statement, it can be concluded that transaction time depends on the type of blockchain, the size of the network, and the applications that run on that network. In other words, different applications will produce different transaction times even though they are built with the same blockchain technology.

Hence, to determine how the work (i.e., type of transactions) will affect transaction time in private Ethereum and Hyperledger networks, it is necessary to know in advance the fundamental work processes expected for the blockchain and Hyperledger. To provide the necessary background for these determinations, this chapter focuses on the technical sides of the Ethereum blockchain and IBM Hyperledger. Details about how the consensus mechanism works on Ethereum and Hyperledger, how a smart contract works, and how Hyperledger caliper supports the benchmarking are also provided in this chapter.

A. BLOCKCHAIN

The decade from 2010 to 2020 has seen the rapid emergence of cryptocurrency blockchain-based technology. Bitcoin, an early milestone in the implementation of this technology for the economic sector, was initiated by Satoshi Nakamoto, the launcher of the first bitcoin block, and this block is known as the Genesis block of bitcoin. In a paper on bitcoin history [3], the author mentioned that the development of bitcoin has been growing slowly, beginning with a deal that became famous on May 22, 2010. The purchase of two

slices of Papa John's pizza using Bitcoin marked the first real-world bitcoin transaction. That day is remembered as "Bitcoin Pizza Day" [3].

Blockchain is a database distributed from the general ledger transaction records to all parties involved in a network—each transaction verified by all parties involved in the system. Once every information input on every blockchain is made, the information can never be erased. Dannen [4], in his book *Introducing Ethereum and Solidity*, mentioned that "the blockchain contains a certain and verifiable record of every single transaction ever made." Yasaweerasinghelage et al. [2], however, found that blockchain technology has significant limitations, including slow transaction time. The authors mentioned that "on Nakamoto consensus (longest chain wins) blockchains, it may take seconds (Ethereum) or minutes (Bitcoin) for a transaction to be included in a block. The latency for the initial inclusion of a transaction in a blockchain is higher than in traditional systems, and a large number of confirmation blocks will multiply this delay."

Currently, bitcoin is only one of the products that use the blockchain concept. After the emergence of bitcoin, the cryptocurrency world has been increasing with the rise of various cryptocurrencies that use the basic technology of blockchain. Rauchs and Hileman [5], in their article, mentioned that, "there are hundreds of cryptocurrencies with market values that are being traded and thousands of cryptocurrencies that have existed at some point." Some cryptocurrencies that exist today are Litecoin, Altcoin, Ether, and many more.

The focus of this research, however, is not on cryptocurrencies; instead, we focus on blockchain as a supply chain application. With the Ethereum virtual machine, the blockchain platform can be utilized for non-cryptocurrency applications. The basic blockchain platform is shown in Figure 1.

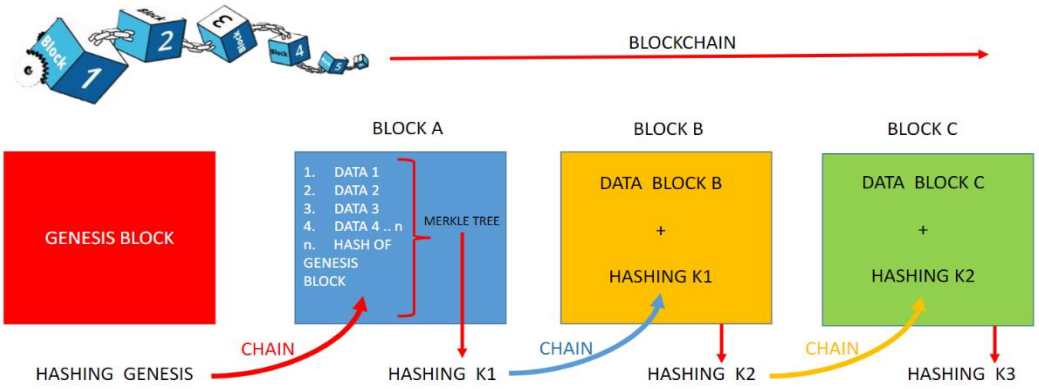

Figure 1. Basic Blockchain Platform

1. Ethereum Overview

Dannen [4] uses the term Ethereum "to refer to three distinct things: the Ethereum protocol, the Ethereum network created by computers using the protocol, and the Ethereum project funding development of the aforementioned two." In order for the system to be truly decentralized, a comprehensive computer network is needed and facilitates the decentralization function. Early in the emergence of bitcoin, the only blockchain network available was the bitcoin network, and its purpose was limited to the cryptocurrency function. If we want to create another framework with features other than cryptocurrency, we need a new system and a new algorithm with a decentralized peer-to-peer function. The first network that emerged for purposes other than cryptocurrency was Ethereum; this network was opened to the public in 2014 by Vitalik Buterin. Ethereum is a "Do It Yourself" platform for decentralized programs, more commonly known as D-apps (decentralized applications). Ethereum's goal is to truly decentralize the Internet.

2. Blockchain Building Blocks

Blockchain is a complex technology consisting of various branches of computer science such as cryptography, Hashing, and Merkle tree. Next, we discuss the basic concepts related to blockchains and supporting blockchain technology as one integrated technology. The next section begins by describing how each block on the blockchain is created.

a. Blocks

Suppose that we assume that a blockchain is a book. Using this metaphor, every page is a block, which is a collection of records of information that has been confirmed by the author and by the publisher. Each block in the blockchain has a unique identity, consisting of a header and a body. This metaphor is shown in Figure 2.

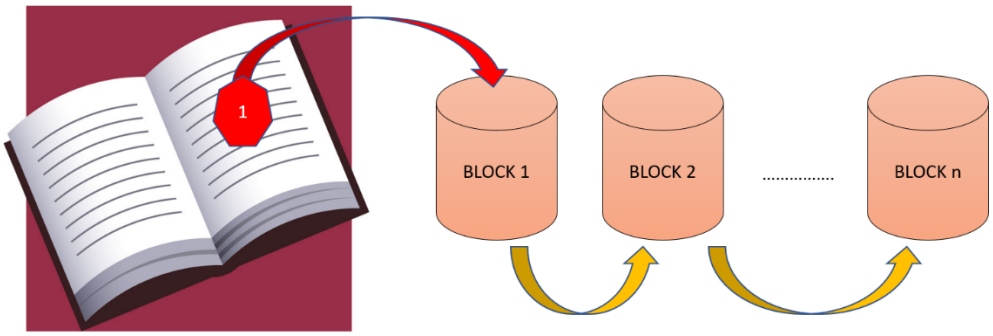

Figure 2. Blocks on a Blockchain

The basic structure of a block in a blockchain consists of four sections, and the first section is the block size. Block size is the size of a block containing the hashes of several transactions in a block and expressed in bytes. The second section is the block header. The third section is the transaction counter, which shows how many transactions there are in a particular block. Then, the last section contains the transactions themselves—transactions that are being processed within this block.

The block header is formed of six fields. The first field is the version, and the version is the number to track the serial number of the updated blockchain software. The next field is the previous block hash, and this section contains the calculation of the Merkle tree of the last block. This section will have a relationship with the next field. The third field is a Merkle tree hash. In his thesis, Merkle [6] explained that "Merkle tree structure helps to verify the consistency of data content." Next, the fourth field is a timestamp. This section contains the estimated time when making a block. In the fifth field, the difficulty target is a proof-of-work algorithm for this block. And the last field is a nonce. Carlozo [7],

in his journal article, mentioned that "nonce is a fundamental concept in cryptography. The nonce is essentially a random number count application used for a proof-of-work algorithm, a random number map which helps to prove the algorithm.." The basic block structure of a blockchain is shown in Figure 3.

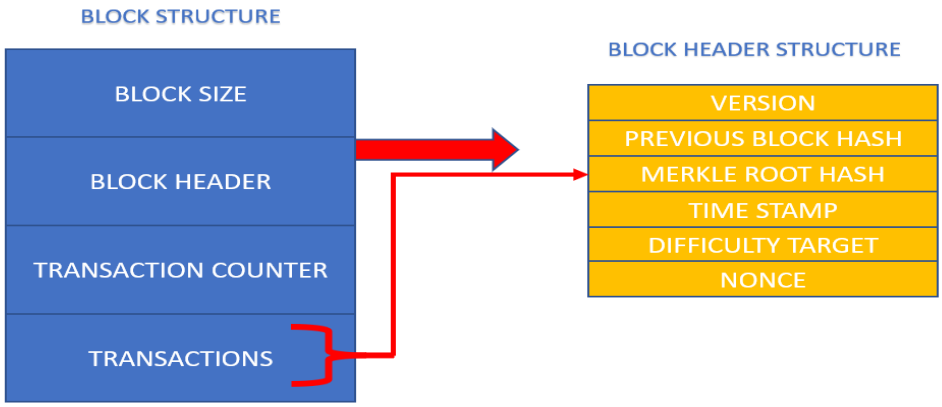

Figure 3. Block Structure of a Blockchain

For illustrating the work of blockchain, it is helpful to use an example. Assume there is a network of two people who want to move money to one another. Assume at the beginning of the process, there is a first block known as the Genesis block, and it has an initial value of five dollars. Then, person A will send money to person B, which creates a record of the transaction in a transaction record. These notes are called ledgers. The ledger consists of several columns. The first column is assumed as the timestamp or the date and time of the transaction. Then, the next column contains details, credit, debit, and balance. Alternatively, it can also include other kinds of information following the purpose of a note.

Now, assume that person A will send money back to person B using the blockchain network. Requests made by person A are broadcasted to a peer-to-peer network consisting of several computers called nodes, which are spread all over the world. When a transaction is transmitted to all nodes, each node will verify the transaction using a specific algorithm that is already known by all nodes. During the validation process, new transactions will be

merged with other transactions that came before, and combined in a pool, called the memory pool. Each transaction validated from the memory pool will form a new block to be merged into the ledger and added to the existing blockchain. In a recent paper, Bowden, Keeler, Krzesinski, and Taylor [5] summarized this process as "a block is a list of transactions, together with metadata including the current time and a reference to the most recent previous block in the blockchain (hence the name blockchain)." Then when a new block is added to the blockchain, the transaction is considered complete.

In the process of inserting a new block into a blockchain, an additional process is needed, known as mining. The blocks created in the previous process from the memory pool must be included in the blockchain, but the node that has the right to insert these blocks into the blockchain must be verified not to be a malicious node. Based on that need, the blockchain has a mechanism to compete for the right to include the block in the blockchain sequence. The winning node has the right and an obligation to insert the new block into the blockchain sequence and get a reward of several bitcoins, or Ether on the Ethereum blockchain. Bowden et al., in their article, also described that "mining is a race between all the Bitcoin miners to find a valid block to append to the blockchain" [5]. With this mechanism, malicious nodes will naturally be selected, because the mining process requires a great effort. Significant computing resources are required to win the mining competition, and malicious nodes will not use large resources to compete for something that they are not likely to win. This method is what we call consensus, and it is described in more detail later in this chapter. Furthermore, early on the consensus algorithm on Ethereum was called a proof-of-work (PoW) process, which has subsequently been replaced by the proof-of-stake process. The process of creating a block in a blockchain is shown in Figure 4.

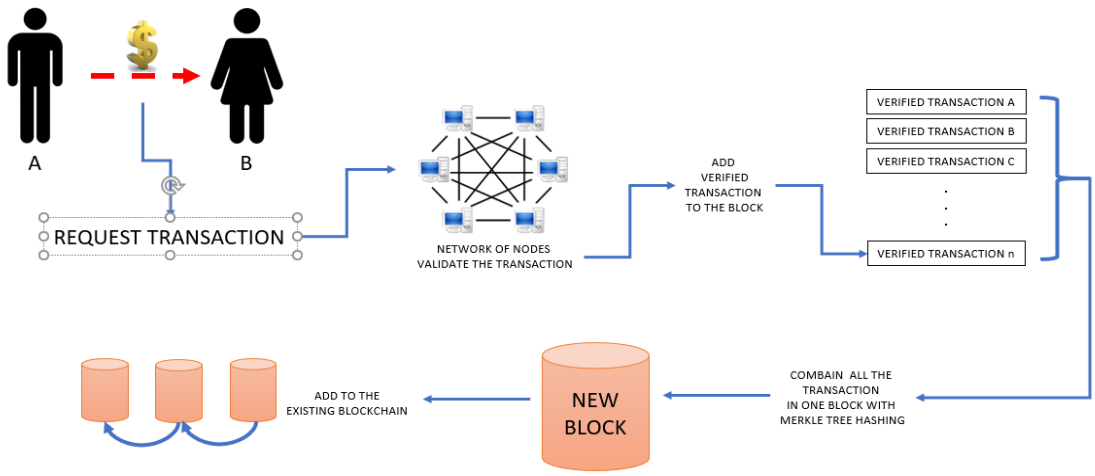

Figure 4. Process of Creating a Block

The mining process in each network will undoubtedly need time to get a consensus. Each node must do a proof-of-work process with a certain level of difficulty. Starting from the consensus process to forming new blocks in a sequence of the pre-existing blockchain, it will require a time value. One online source [9] mentioned that "the length of time for mining is called block times. Block time defines the time it takes to mine a block. In bitcoin, the expected block time is 10 minutes, while in Ethereum it is between 10 to 19 seconds."

Block time in bitcoin and Ethereum can be defined with two types of time; expected block time and average block time. Siriwardena [6] elaborated that

> the expected block time is set at a constant value to make sure miners cannot impact the security of the network by adding more computational power. The average block time of the network is evaluated after *n* number of blocks.

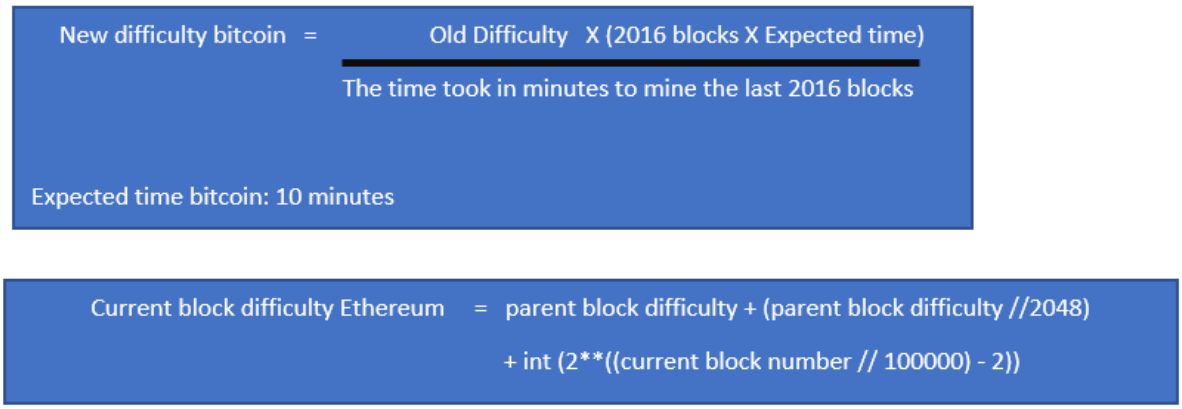

Figure 5. Difficulty Formula for Bitcoin and Ethereum. Source: [9].

As previously noted, the expected block time on Ethereum is 10–19 seconds. If the latest block is mined with time below the expected block time, then the next block's difficulty will be increased. If, however, the block time is between 10 and 19 seconds, there is no need for changes in the next block's difficulty. The formula for the relationship between block time and difficulty is shown in Figure 6.

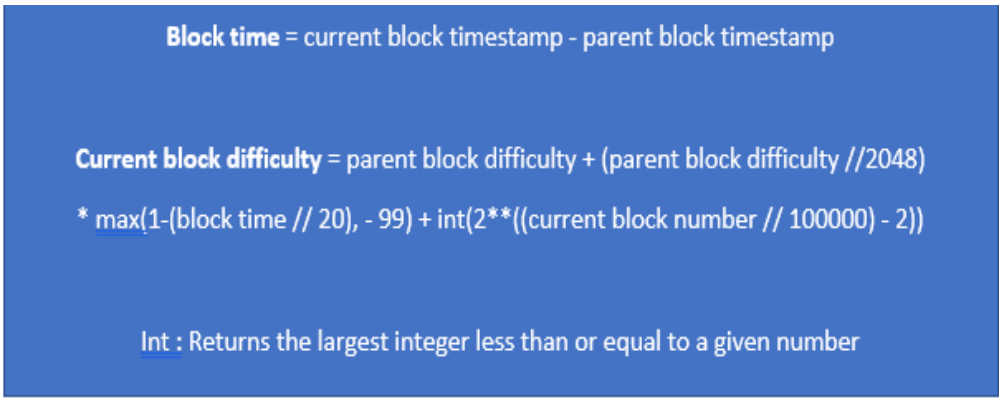

Figure 6. Relationship between Block Difficulty and Block Time. Source: [9].

Having discussed difficulty and block time, we must now consider network latency in Ethereum. Network latency has a strong influence on how long a user waits for each transaction made to enter the blockchain. In general, people understand the concept of

transaction time by visualizing a vehicle's motion from point A to point B, and how long the move will take. Nevertheless, this is different on the blockchain. The movement is periodic rather than continuous, as illustrated by the example in Figure 7, where periods are described as refresh rates. Then, the question is how often the blockchain refreshes or updates itself.

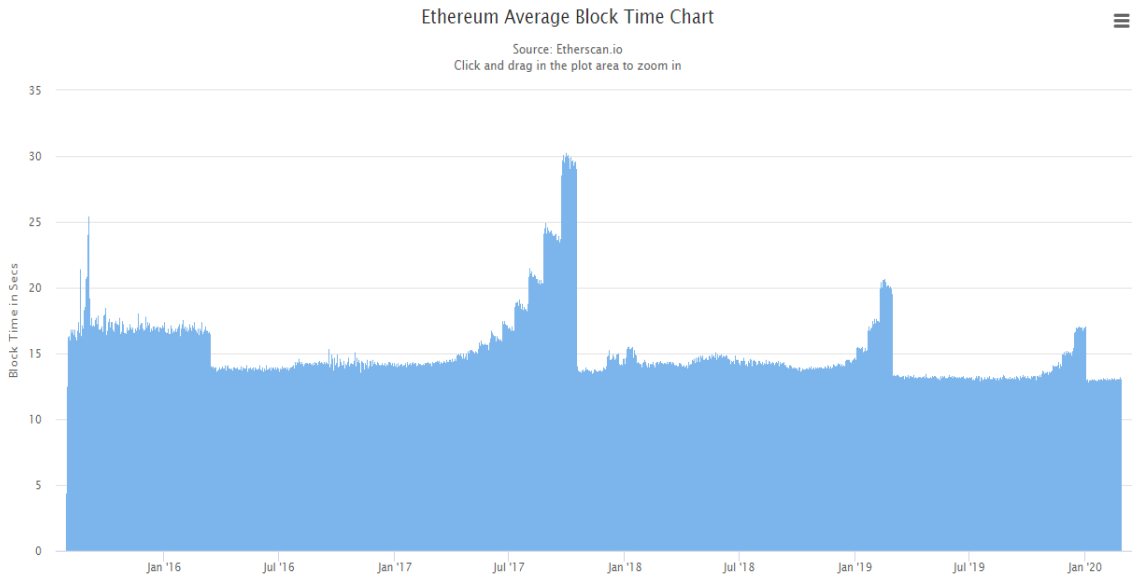

Figure 7. Ethereum Average Block Time. Source: [9].

According to fundamental theory about blockchains published by medium.com, bandwidth is the same as transactions per second (TPS); TPS has two variables, block size and block time. The block size is limits how big bitcoin blocks can be. Transactions per second are the same as block size multiplied by block time. The maximum Ethereum block size based on Etherscan.io is between 20 and 30 kb in size. The factor determining the size of this block is how many units or gases are spent per block, where gases are the fees given to register transactions on the Ethereum network. An Ethereum average block size chart is shown in Figure 8.

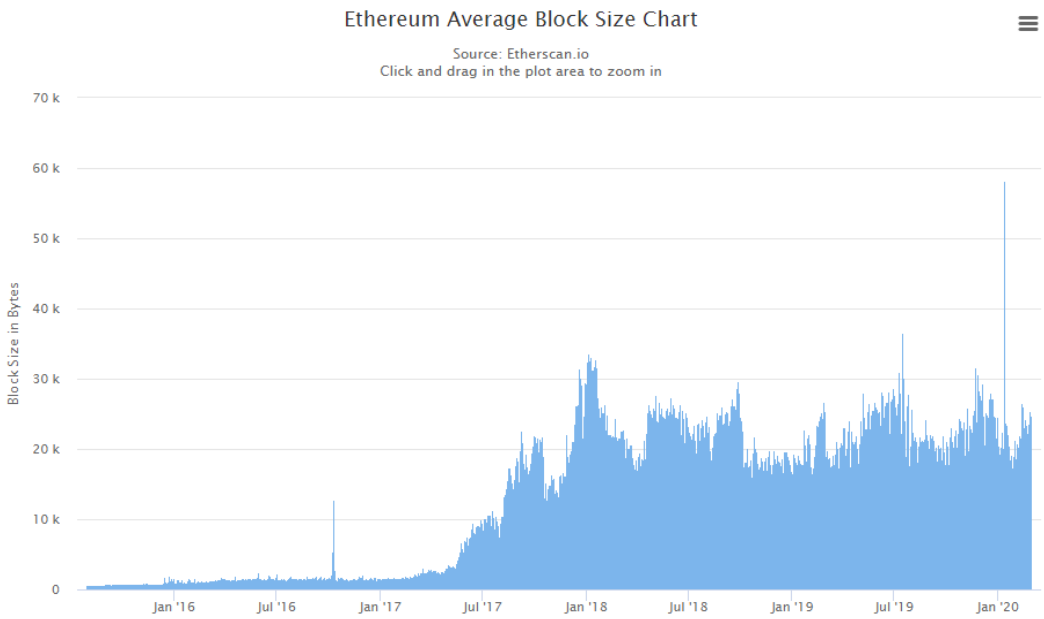

Figure 8. Ethereum Average Block Size. Source: [9].

b. Smart Contracts

Smart contract is the most innovative aspect of blockchain technology, serving as a bridge between the user and the ledger system on the network. Smart contracts help users in self-executing their agreement within the blockchain system itself. The smart contract translates business logic, as stated in the business cooperation agreement, into automatic computer logic. Hence, every transaction within the blockchain network no longer requires a third party, such as a broker, lawyer, bank, or another intermediary. Users can directly generate new contracts registered with the blockchain.

The Ethereum programming language, which is used to build a smart contract, is a programming logic that functions to run D-app. For example, in real life, a contract is a legal device containing "If" and "then" statements. As an illustration, we can take a real-life example, if we sign a lease contract to rent a house and agree to pay the owner $ 3000 on a specific date, the owner of the contracted house will allow us to stay when the lease payments have transferred to the homeowner's account. The transaction will run automatically without the need for both parties to verify mutually. A smart contract will do everything automatically. An example of an Ethereum smart contract is shown in Figure 9.

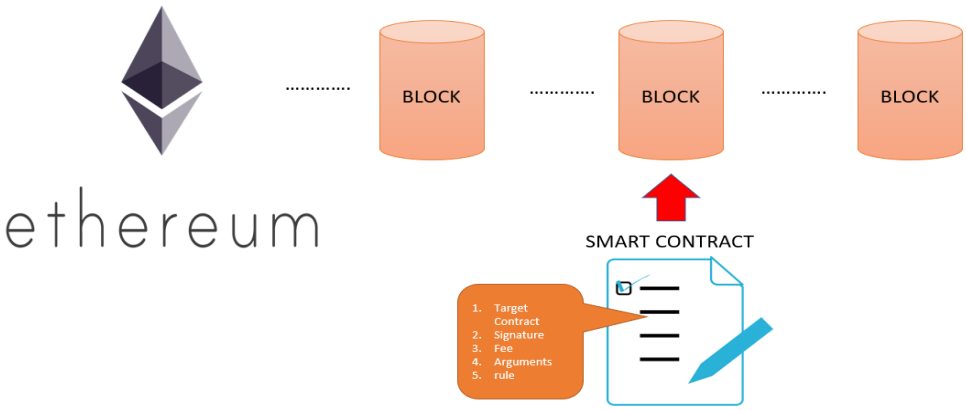

Figure 9. Smart Contract

c. Peer-to-Peer Networks

On decentralized networks, the network follows the client and server hierarchy. The server has a position at the top of the level; the server shares information needed and requested by the client. A decentralized network is very different from peer-to-peer networks in which each computer is connected to the Ethereum network as a node that has an equal position. Each node will prepare its resources such as disk storage, processing power, and network bandwidth to continue the joint connection, without the aid of central coordination. The main difference between the two systems is the primary point of authority. The peer-to-peer network has no central authority.

A peer-to-peer network, sometimes referred to as a P2P network, is critical for blockchain technology. There is no central point of storage on this type of network, and therefore, no governing party. All the information that exists on the system is continuously recorded and transferred between the nodes on the network. All nodes store multiple identical copies of the network. A peer-to-peer network is a distributed network that stores and transfers data without a central server. Similarly, a blockchain has no central point of storage; this condition makes the information on a system far less vulnerable to being hacked, exploited, or lost.

A peer-to-peer network on a blockchain is shown in Figure 10.

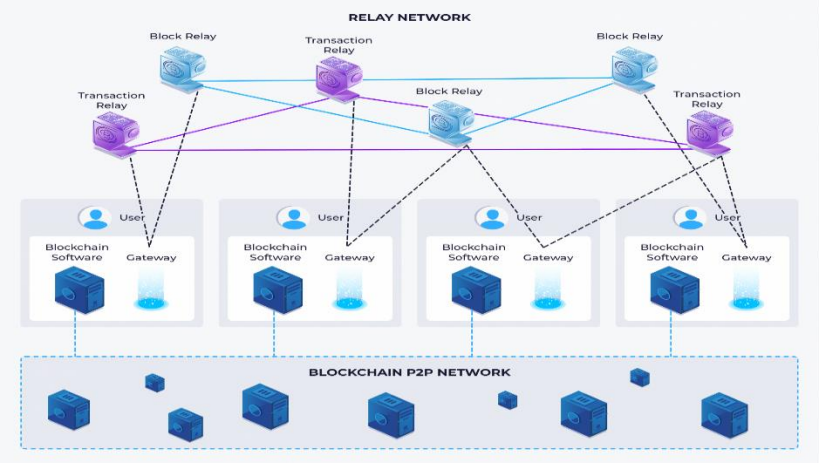

Figure 10. Peer-to-Peer Network on a BlockChain. Source: [9].

d. Consensus

To clearly illustrate what consensus is, it is better to use example cases again. As noted previously, blockchain works on a peer-to-peer network where each node stores the same transaction history on the blockchain. Suppose we have five nodes, each of which has the same database of all transactions that occur in the network. When a new block with the latest transactions must be entered into the blockchain, which node will have the responsibility to enter the block into the blockchain? How is it possible to know for sure whether each node is a non-malicious node or not? What if one of the nodes is a malicious node? Hence, we need a mechanism called the consensus algorithm that can ensure the latest block can safely, and by the legitimate node, enter the blockchain. In other words, the system has guaranteed the integrity of the data.

The node that will insert the latest block into the blockchain will be determined by consensus. There are many examples of consensus algorithms; the most famous is the proof of work (PoW) mentioned earlier. Using PoW, all nodes use all their computational resources to do a calculation to solve a mathematical puzzle. Each node that succeeds in completing the puzzle will be the winner and can add the latest block to the blockchain. In bitcoin, every node that does the calculation is called a miner, and these nodes will get a reward each time they complete a mathematical puzzle calculation. Because they have devoted all their abilities and resources to achieve a mathematical calculation, the nodes

prove that they are not malicious. Therefore, the consensus algorithm protects against the vulnerability that possibly arises from a malicious node.

In Ethereum, there is another consensus algorithm named proof of stake (PoS); this is an update of the PoW algorithm. In PoW, a node must use existing resources to win a mathematical calculation. With PoS, by contrast, the node only needs to invest a certain amount of money (gas) in winning the consensus competition. This algorithm can also serve as a filter against malicious nodes because malicious nodes will not dare to invest specific values if they are not sure that they will appropriately and quickly win a consensus competition. Figure 11 provides a high-level comparison of the PoW and PoS methods.

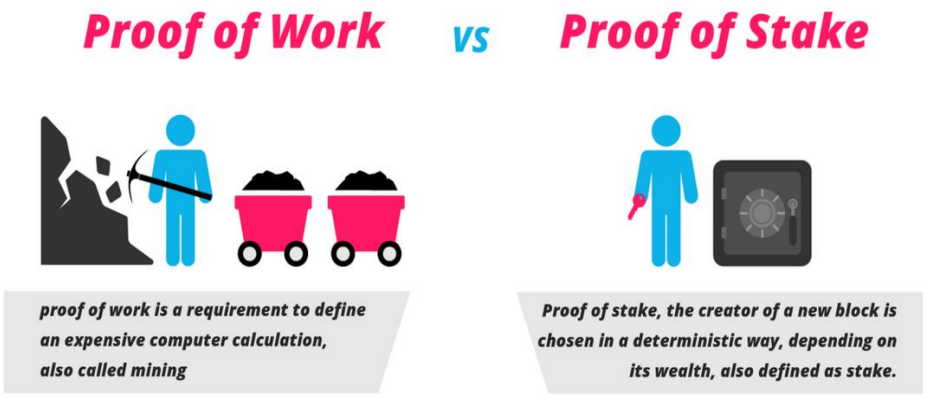

Figure 11. Proof of Work versus Proof of Stake. Source: [8].

B. IBM HYPERLEDGER

As we already know, cryptocurrency is an application that applies the underlying technology of the blockchain. Ethereum represents other interests, focusing not only on cryptocurrency but also on distributed applications based on blockchain technology. Nevertheless, both products are examples of public blockchains.

In 2016, the enterprise blockchain known as Hyperledger was launched. To understand why Hyperledger came about, it is necessary to discuss in detail what a Hyperledger is and how it differs fundamentally from Ethereum. In this section, we discuss Hyperledger in detail as well as the aspects of latency and block time as compared to the

previously described Ethereum technology. This discussion can be used by the United States Navy, in particular, for comparison data to determine which background technology is better suited for ANUSD.

1. Overview of Hyperledger

To discuss Hyperledger, first we need to know the root difference between this technology and Ethereum technology. In Ethereum, when a transaction occurs, the entire transaction data will be sent to all nodes as part of the consensus process. Even though the data is encrypted, there is a possibility that reverse engineering can occur to find out the purpose of the transaction, which can be easily peeked at by all nodes. In businesses such as banks, companies with valuable intellectual property, or factories, this concern becomes crucial; privacy is vital. Sometimes other companies engaged in the same field must maintain the confidentiality of their activities from their peers. For example, if in a fruit selling business circle, sometimes producer A gives a special discount to customer X, producer A may not want other customers or producers to know this policy. Privacy and policies therefore must be maintained in the transaction process. Moreover, there is also an issue of competitors. As Shah [10] mentioned in his book *Blockchain for Business with Hyperledger Fabric*, "all banks basically compete with each other in order to create a better market for themselves. Thus, they prefer to keep their transaction details, strategies, and policies confidential to ensure that others do not leverage their competitive advantage."

One fundamental difference between Ethereum and Hyperledger is that all nodes that have joined in a Hyperledger network know each other. Any node connected to the network must have permission to use specific methods, which provides another level of security on the network.

In this thesis, we focus more on Hyperledger Fabric. Hyperledger Fabric is the framework underpinning the IBM Blockchain platform, which is an open-source platform. The following section discusses the Hyperledger Fabric components.

2. Components and Consensus Protocols of Hyperledger Fabric

Hyperledger has another advantage over Ethereum, pluggable consensus protocols, which are consensus platforms that can be changed to follow an application system's interests. Hyperledger Fabric users can modify a consensus protocol to support specific goals.

The next equally important component is the Apache Kafka and Zookeeper tools. Based on Hyperledger documentation officially published by the Hyperledger Fabric website, we learn that "Hyperledger is high-throughput and fault-tolerant distributed messaging system" [8]. Its ordered uses Apache Kafka and Zookeeper as tools to reach a consensus, order transactions, and put them in a block and send them to every peer in the Hyperledger Fabric network organization.

A peer is a member of the Hyperledger Fabric organization. For example, if Example.com Company has administration, human relations, and finance departments, then the peers are the staff members of each of these departments. Hence, we can identify peer-1 in administration.example.com as the first staff member in the administration department at Example.com Company. These peers have the task of maintaining the chain code, endorsing transactions, and validating the transactions and blocks. Peers themselves have anchor peers, endorser peers, and validating peers. Each organization must have anchor peers whose job is to communicate with other nodes outside the organization while remaining on the same channel.

Furthermore, every organization must have at least one endorser peer, whose job is to accept transactions originating from client applications. Validating peers work like peer endorsers, and sometimes endorser peers can become validating peers. Their job is to validate whether a transaction can be entered into the ledger.

The next node that must exist in every organization is the orderer node. The orderer node is the heart of the Hyperledger Fabric system. This node will connect each member on the network. The main task of this node is to order transactions originating from the client application to the ledger.

The certificate authority (CA) node is another significant node for the Hyperledger Fabric process. The task of this node is as the certified infrastructure of a Hyperledger network.

Other tools needed by the Hyperledger Fabric network are the command line interface (CLI) and CouchDB. CLI is a container docker with several tasks, such as creating a channel, installing chain code, and upgrading the encoding. CouchDB is a database needed in each node as a place to store various key-values and ledger information. Each peer has one CouchDB for data storage. An example of a Hyperledger Fabric network with two organizations is shown in Figure 12.

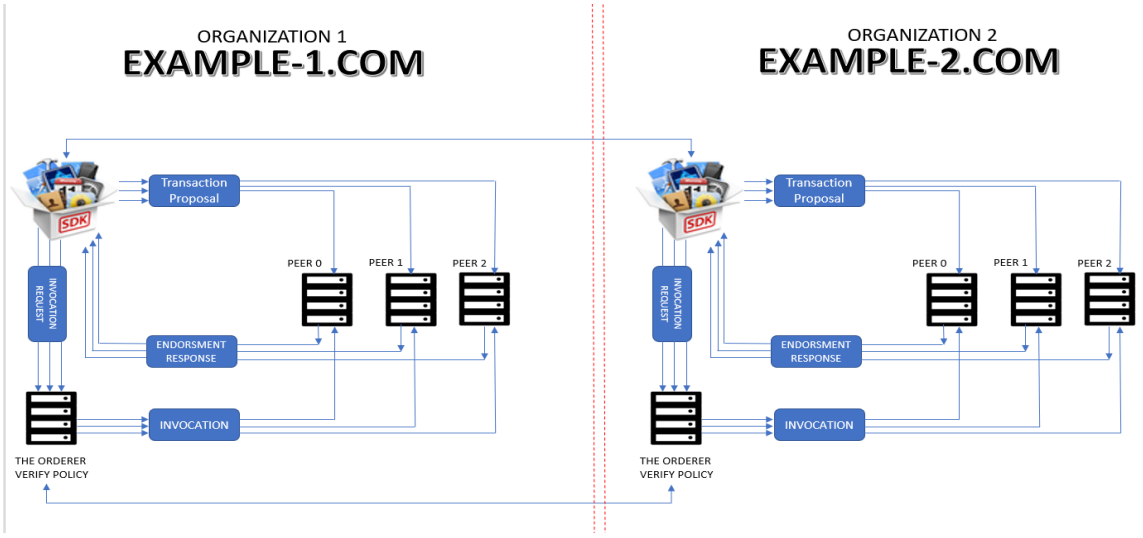

Figure 12. Example of a Hyperledger Fabric Network with Two Organizations

From the picture in Figure 12, we have two organizations on the same Hyperledger Fabric network, and each organization has three peers and one orderer node. It appears that each organization also has an input line from a software development kit (SDK). Each SDK has a responsibility to connect the client application with the Hyperledger network. To make it easier to explain how the consensus process works on a Hyperledger Fabric network, we should give an example. Assume that if the client will change the ownership data of an asset registered in the previous blockchain, the client will input a transaction that

comes from the SDK. Then, every transaction will get a certificate from the CA node. Next, the transaction is transformed into a transaction proposal. It will then be sent to one of the peers (or all peers) depending on the policies made earlier. Transaction proposals received by peers will be simulated to be included in the current ledger, but the existing ledger itself will not be changed because the transaction proposal is only a simulation.

The result of the simulation is a key called the read-write set. The principle from this process is that the peer will accept every transaction proposal from the client and sign it cryptographically. Afterward, the results will be sent back from the peer to the SDK; after that, the SDK will receive all the results from the peer recommendations and enter that information as an invocation request. Then, the invocation request packet will be sent to the orderer node. The orderer node, signature verification, policy verification, and other verifications are used as a filter. The orderer node verification results will be invoked in the ledger and sent to each peer to be updated on the new ledger. Whether the results have followed the policy or not will also be entered into the ledger, so that later the client can see all transactions that have either failed or succeeded from the ledger for review purposes.

The fundamental difference between the consensus process that occurs on the Ethereum blockchain and the one on the Hyperledger blockchain is the absence of a consensus race. Hyperledger does not have a miner whose job is to verify and insert new blocks in the ledger (blockchain). The function has already been taken over by the orderer node. Moreover, Hyperledger has no incentive system.

3. Block Time on Hyperledger Fabric

IBM published Hyperledger Fabric as an open-source project in 2015; Fabric is one of the available Hyperledger frameworks. Some versions of Hyperledger currently available are version 0.6, version 1.0.0, version 1.2.0, version 1.3.0, and version 1.4.0.

During the course of his research, this author consulted a reference comparing the block times for Hyperledger Fabric V1.0 and V0.6 using a smart contract application in a simple money transfer application. The source mentioned that “The execution times for

Fabric v1.0 are better than the execution times for Fabric v0.6" [9]. A sample graph from that work comparing transactions per second is shown in Figure 13.

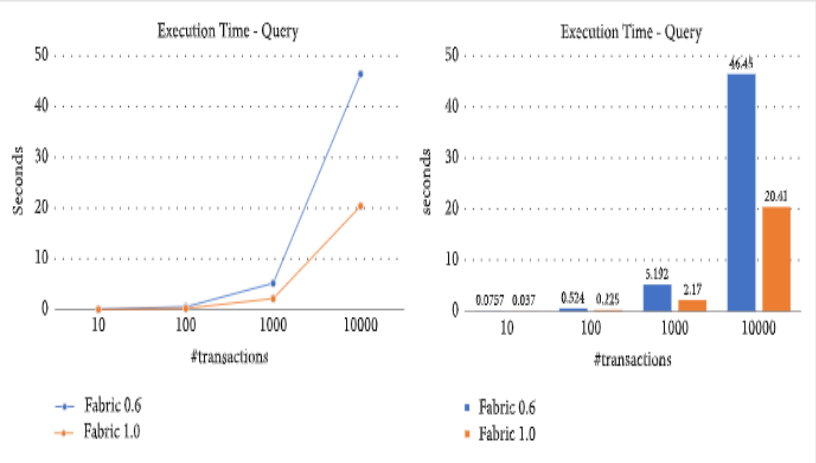

Figure 13. Transactions per Second Graph. Source: [9].

In this thesis, we use Hyperledger Fabric version 1.0 for our research comparison with the Ethereum blockchain. We examine the number of transactions per second of the two systems by implementing a smart contract application. The application applied to both systems is the ANUSD, which is a decentralized application providing a smart contract functionality for software distribution that will communicate with the U.S. Navy's blockchain network.

C. DECENTRALIZED APPLICATIONS

A decentralized application, or D-apps, is an application layer. It is directly in contact with the client, while the blockchain comprises a protocol layer below the application layer hierarchically. D-apps serves as a bridge between the client and the Ethereum blockchain or Hyperledger protocol. D-apps is an open-source programming code that can be accessed and used by others. Due to the decentralized nature of D-apps, it enables all data processed to be distributed to all member nodes on peer-to-peer networks.

The D-apps used in this research is ANUSD, as just mentioned. We make simple D-apps specifically for the Ethereum and the Hyperledger Fabric networks. Both D-apps

have similar functions but for two different networks. They are then tested for block time and scalability in each system.

To provide a basic understanding of making simple D-apps, it is necessary to describe the differences of each D-apps in Ethereum and Hyperledger Fabric.

The decentralized application used in the Hyperledger Fabric is called the Fabric SDK, which is an outer part of the Hyperledger network having many essential functions. The SDK can be made from various programming languages, such as Node.JS, JAVA, and so on.

The function of the Fabric SDK is the same as it is for D-apps on an Ethereum blockchain. It is the link between the client and the blockchain network, but the Fabric SDK does not have a consensus reward system like Ethereum. The orderer node solves the consensus. Therefore, different methods and processes are needed to test block time and network latency in the ANUSD application for the Hyperledger Fabric network. The testing process for measuring block time and network latency is assisted by Hyperledger Caliper, a tool that enables the benchmarking process.

D. HYPERLEDGER CALIPER

We can use a caliper to measure the performance of the implementation of the blockchain. Measurement of performance is essential for every user who applies blockchain technology. Therefore, at this time, the Hyperledger Caliper is used in our thesis research.

According to a 2018 Linux Foundation blog about performance analysis of Hyperledger, “Hyperledger Caliper will produce reports containing several performance indicators, such as TPS (Transactions per Second), transaction latency, resource utilization, etc.” [9].

Caliper’s benchmarking criteria so far include success rate, transaction latency, and resource consumption. In our research, we focus only on transaction latency, highlighting the extent of the ANUSD application latency in the Ethereum and Hyperledger Fabric networks, respectively**.**

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN

This chapter describes the design of the Ethereum blockchain and IBM Hyperledger frameworks and also the ANUSD application for each framework. The same guidance is necessary to determine the type and number of organizations and the number of nodes involved in each framework. Despite differences in the frameworks of the Ethereum and IBM Hyperledger, they must support the ANUSD application so that it has the same functionality and can support the same number of node organizations to generate valid testing and make an accurate comparison of block time and scalability on both frameworks.

A. ANUSD IN DECENTRALIZED APPLICATIONS

The ANUSD is a supply chain application capable of sharing software automatically from the producer side through various intermediary nodes until it is accepted by the client, using blockchain technology. This application is intended for use by the U.S. Navy, as the name suggests, ANUSD or Autonomous Navy Unclassified Software Distribution.

1. Goal Hierarchy

The goal hierarchy of the ANUSD application is to be the main application applied to the Ethereum network. It functions as a supply chain application whose job is to facilitate the role of the supply chain. For our purposes, the ANUSD application will run a randomized scenario of sending data/software from one point to another to research the latency of the Ethereum network. The ANUSD application developed in this thesis is a simple prototype; various complex functions that may exist in the real supply chain system were not applied to the ANUSD here. The development of the ANUSD application for this research followed the organizational architecture depicted in Figure 15, and a few essential functions were used. These functions included signing/validation performed by each organization, and the transaction time measurement features. This condition relates to the complexity and scope of this thesis. The thesis aims to describe differences or similarities

in terms of transaction time and scalability resulting from the implementation of the ANUSD application's essential functions on two different blockchain frameworks.

The ANUSD is a supply chain application that adopts blockchain technology, which raises the security factor as a main issue. Product security focuses not only on anti-theft protection but also on the protection of material sources that make up a product, meaning that a product is expected to be genuinely safe, from the beginning of the manufacturing process to distribution to clients. Security can be achieved by keeping a log of any transaction securely. Keeping a record of every change to an object or software in ANUSD is required for security monitoring.

Generally, supply chains are identical for both hardware and software products. Manufacturers need to ensure that all raw materials of a hardware or software product come from reliable sources, ensure consistent outcomes, provide products with outstanding protection during storage and transit, and give a warranty and validation upon delivery.

2. Environmental Model

The environmental model of the ANUSD application refers to the traditional environmental supply chain but has additional features in the form of blockchain technology. The blockchain implementation will increase the level of trust of its users and grow due to a trusted security guarantee. Also, with the blockchain, the environmental supply chain model will have a feature of automation. Every transaction on the blockchain network will be done automatically with the help of a smart contract, which is explained further in subsequent sections.

The essence of a blockchain system is trust and automation. Any transaction in the order will be recorded in the ledger, thus reducing the threat of tampering from each transaction's data. The threat is minimized because every transaction is notified and approved by all parties in the network. The only gap that can change transaction data is a 51-percent attack, i.e., the condition in which at least 51 percent of the network members approve the change. Nevertheless, such a gap is tiny in a network with thousands to

millions of members, as the attacker will have to tamper with 51 percent of the network members simultaneously and within a specific time limit.

Chang [14], the global blockchain lead for distribution and the industrial market at IBM Corp, as reported in Block Chain Tech News, gave a presentation at the 2019 National Restaurant Show on how the traditional food supply chain provides limited activity visibility to its participants. The interactions between the grower, manufacturer, transporter, and retailer are not streamlined. On a blockchain, however, the participants are all part of one streamlined journey. He said that "The data that's sitting on the blockchain can be trusted; a blockchain is going to help your business become more automated. A blockchain ecosystem can be seen as a social network on which the participants can communicate with one another in real time" [10]. This condition also applies to the supply chain system for ANUSD distribution software, all part of one streamlined journey. Figure 14 illustrates the comparison between traditional supply chains and blockchain technology-based supply chains, which is applicable to that of the software distribution on a traditional supply chain in comparison to the ANUSD supply chain that we build.

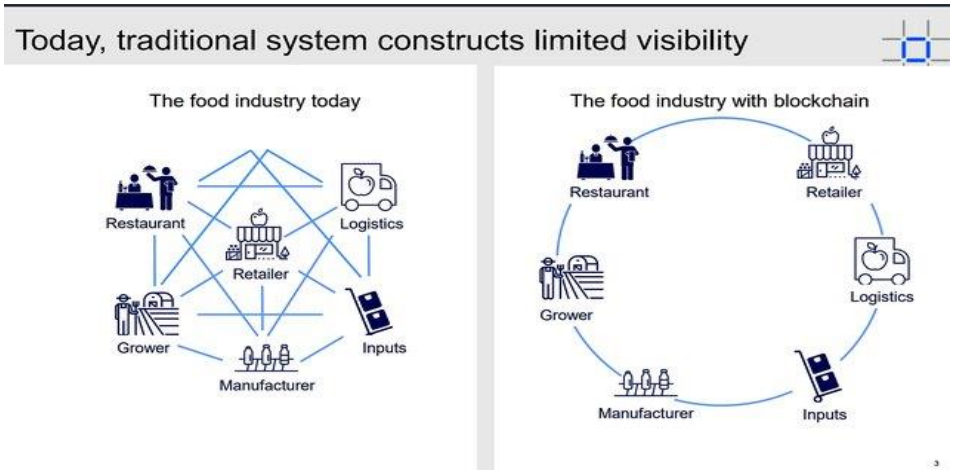

Figure 14. Traditional versus Today's Food Supply Chain. Source: [10].

The ANUSD software was developed based on the medical blockchain architecture described in "A Novel EMR Integrity Management Based on a Medical Blockchain Platform in Hospital" [11]. This article was used as the basis for ANUSD architecture

development in our research because this system has the same function, to transfer data of different sizes from each node to another in the network.

Our ANUSD architecture has the same architecture as the blockchain-based E-Health architecture but with a different designation. The essence of the ANUSD application is the transfer of software from the manufacturer to the client. The ANUSD application scenario is shown in Figure 15.

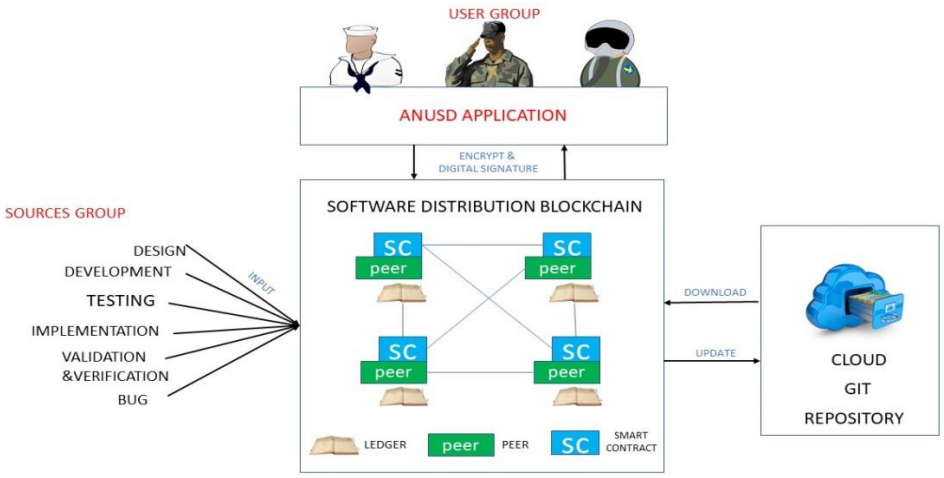

Figure 15. ANUSD Architecture

As shown in Figure 15, the ANUSD network has validating peers, and every node keeps a file copy of the ledger to maintain consistency of the distributed ledger. The ledger itself contains immutable transaction records. In this thesis, peers in the ANUSD system consist of the Peer Client, Peer Developer, and Peer Quality Control. The simple analogy implemented is software delivery from the developer to the client and through the peer quality control. The software product delivered can be returned or forwarded to the previous or next peer, and this is illustrated in Figure 16.

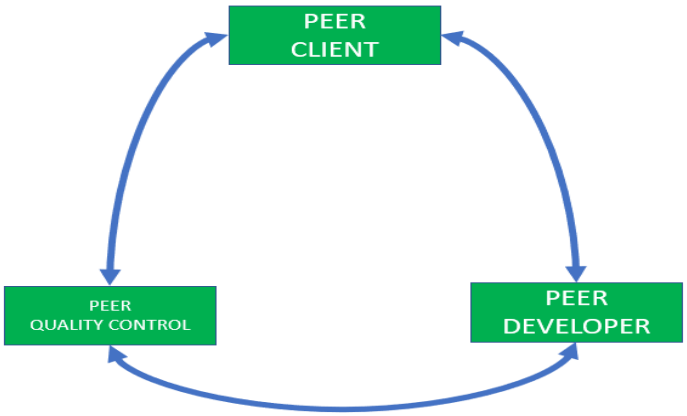

Figure 16. ANUSD Data Flow

Based on Figure 16, the software delivery data flow for the ANUSD application begins in the following order: the client orders an application from a developer, the developer reviews or approves the client's request. Upon completion, the developer has the peer quality control perform testing on the application, and if the application passes the test, the peer quality control delivers it to the client. If the application does not pass the test, the peer quality control returns it to the developer. Each transaction just described is recorded and verified by the parties in the blockchain system and follows each existing framework's rules.

The ANUSD application provides users with high-level security as they can receive any software sent from authorized and reliable parties that are part of the Department of Defense supply chain. Every recorded change to design, development, testing, implementation, validation, and bugs will be logged in the ledger. The data cannot be deleted by any party, even by someone with the highest authority in the Navy. This condition happens because blockchains are decentralized and do not have a central authority, meaning that all parties in the supply chain have an equal position.

3. Service List

The ANUSD developed in this thesis provides some services. The first service is the transfer of data/software from one node to another; the second service is to limit the

number of data transfer transactions according to the test scenario of 10–10,000 iterations. The third service is to divide the number of transactions made by each node randomly. For example, if we have three nodes involved in 1,000 transactions, then the ANUSD will have a feature to divide the number of transactions into blocks of 333, 333, and 334, or another random number that evenly distributes to each node. All service features are implemented automatically by the ANUSD application. And to facilitate the latency trial, a web-based system is applied.

The ANUSD application for block time and scalability testing in this thesis was developed based on web technology. The web is connected to two different frameworks but produces monitoring displays on the same web sheet to make monitoring the block time and scalability calculation results of the two different frameworks convenient.

This website is connected to the IBM Hyperledger framework and the Ethereum blockchain framework through a smart contract. The ANUSD web application automatically sends every request by a peer via a smart contract to two different frameworks. Then, each framework records the transactions in their respective ledgers based on the applicable rules. During the recording of new transactions into the ledger, an additional process is carried out, i.e., benchmarking from latency and scalability calculations performed by Hyperledger Caliper in each framework. The final process is sending the latest ledger records to each peer simultaneously with a transfer of the latency results (transactions per second) and scalability benchmarking calculations from the Hyperledger Caliper to the ANUSD website, as illustrated in Figure 17.

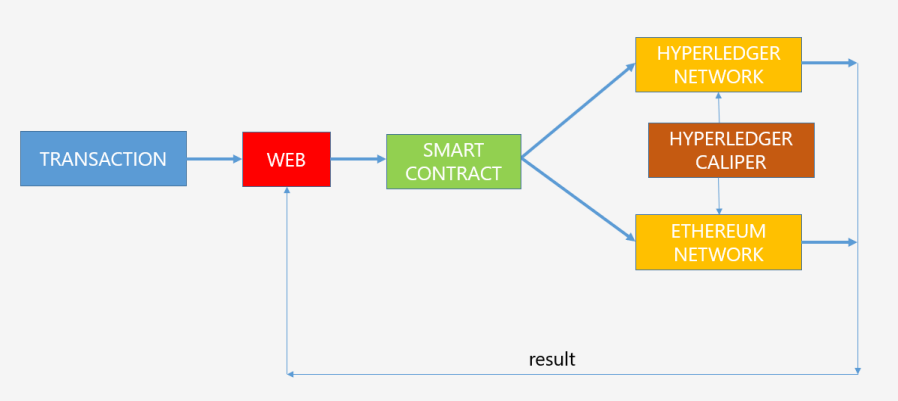

Figure 17. Web-Based ANUSD Data Flow

The ANUSD application website can communicate with the IBM Hyperledger framework and the Ethereum blockchain framework as a smart contract. Each transaction made by a peer has different functions depending on the task of each peer. Smart contracts prepare transaction-related forms that facilitate transactions. One of the smart contracts' primary services is "SEND FILE," which is sending various types of files from each peer to the intended peer and keeping records of the update data to the ledger.

The ANUSD website application is built on a public Virtual Private Cloud (VPC) and is linked with the IBM Hyperledger framework and Ethereum blockchain on other VPCs. The VPC used is the Amazon AWS.

The next process to improve this architecture is describing the designs of the Ethereum blockchain framework and the IBM Hyperledger framework to facilitate this ANUSD application and the design of Hyperledger Caliper that functions as the benchmarking framework. These designs are discussed in the following sections.

B. ETHEREUM BLOCKCHAIN DESIGN

The Ethereum blockchain design in this thesis is explained by several approaches, including the framework approach, network topologies approach, and consensus. These approaches serve as a primary guideline document for the implementation process in the next chapter.

1. Framework

The Ethereum blockchain framework in this thesis is built on three AWS nodes. Each node represents a supply chain organization. The organizations provided to support the ANUSD program are the client organization, developer organization, and quality control organization. In practice, if the U.S. Navy uses this concept in the future, each organization may consist of several peers. For example, a client organization may be composed of peer Marines, warships, educational institutions, and peer staff. The Ethereum organization, however, consists of only two peers at the beginning, but peers can be added up to a total of six, as is explained in detail in the next chapter.

Development of the Ethereum blockchain framework on these three different nodes primarily focuses on how Ethereum nodes work together, and how each node interacts with smart contracts in different nodes. The architecture aims to describe the real conditions in the U.S. Navy environment, where each organization is located in a different location. This condition also illustrates the ways a decentralized application functions in three various organizations without involving a central authority.

This framework deploys a t2.medium server with 3vCPU 4 GB RAM and 8 GB SSD. The operating system used is Linux Ubuntu version 18.04. The three nodes use the same security group that allows TCP 30303; this port is used by default in all three nodes' peering process. The decision to use the t2.medium server was made because, based on experience, in several installations the t2.small does not function properly, particularly when a smart contract is executed for a particular transaction. We found that the t2.medium is more stable to use.

2. Network Topologies

In this thesis, the Ethereum blockchain network topology uses complex network modeling. Every account incorporated into the network is represented as a node, and every transaction that occurs between nodes will be recorded in a blockchain. Any change will be recorded and verified by all nodes in the network.

The number of nodes in the Ethereum blockchain framework is equal to the number of nodes incorporated and interacting. This thesis uses four nodes consisting of three separate nodes, each in different organizations, one miner node, and one benchmarking node. Interactions between nodes are seen as a link. The link describes the interaction of software delivery transactions and verification that occurs between nodes.

The nodes used in the private Ethereum blockchain topology are static. As mentioned by Poon [12] in a Microsoft CSE developer blog, "static Nodes are pre-configured connections which are always maintained and re-connected on disconnects by Ethereum nodes. As static nodes will be connected to directly, no UDP discovery is required". To connect the Ethereum blockchain framework member, the Peer-to-Peer Port will be used on each node (Geth-TX-Public) in each organization. The topology of the three-node organization is illustrated in Figure 18.

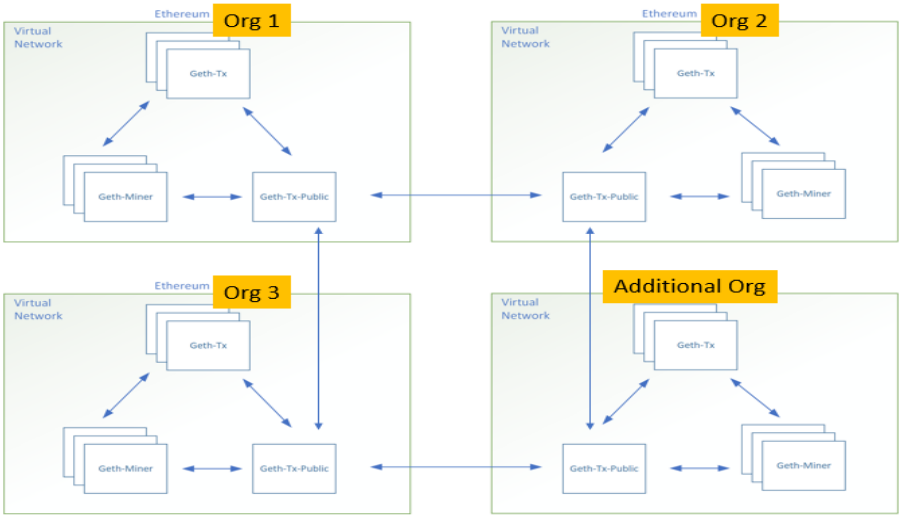

Figure 18. Ethereum Blockchain Topology. Source: [12].

3. Consensus

The initial conceptual model for consensus adopted in the Ethereum blockchain framework is the PoW process. PoW is a security mechanism to protect a network against possible denial-of-service attacks or in-network spam. To prove that a user is a legitimate

user and eligible to join a network requires some kind of PoW from the user. This verification process involves computer processing and computer resources.

As explained in the previous sections, this framework has four nodes, and each node can change into a miner. A miner can solve complex mathematical puzzles, but it needs power resources and processing to do so. When one of the four miners manages to solve a puzzling problem, the miner is entitled to a reward and must enter transaction records into the ledger. After that, the winning miner will broadcast the latest ledger structure to all nodes in the network.

The calculated puzzle is a nonce, one of the requirements to build a new block in the ledger. No one can insert a new block into the ledger without knowing the nonce. Thus, the nonce provides security against the possibility of unauthorized user entry into the blockchain network. Fake users will not use their capabilities and resources for things that remain uncertain. The cost incurred would be higher than the potential rewards.

The technical concept of PoW applied to this framework is the hash-based PoW. This PoW requires a calculation to determine the nonce value. Hash technology is the basis for this calculation. The Merkle hash from the previous block needs to be smaller than the current target block value. Once the nonce value is found, the miner will create a new block and pass it on to the network. The other peer nodes will verify it after receiving the calculation. The calculation carried out in this thesis is related to the winning miner's new block. The calculation is based on block intervals.

In this research, the latency calculated is determined by difficulty. In this case, it is the mining difficulty, which is a rule that determines the degree of difficulty for each miner to discover the eligible hash explained previously. The difficulty setting model applied is difficulty adjustment from the lowest difficulty value (i.e., the easiest) to the highest difficulty value at a certain threshold.

C. IBM HYPERLEDGER DESIGN

The IBM Hyperledger blockchain design in this thesis is explained by several approaches, including the framework approach, network topologies approach, and

consensus. The approaches serve as a primary guideline document for the implementation process in the next chapter.

1. Framework

In this thesis, a Hyperledger framework is developed using multimode organizations. Specifically, we use the Hyperledger Fabric version 1.4LTS, the stable version, which was published in 2016. It is an enterprise-grade and open-source distributed ledger framework that supports business transactions. The Hyperledger Fabric framework is modular and serves as a standard framework for the enterprise blockchain platform. It is modular because each component, such as consensus and membership services, can be installed on the framework using the plug-and-play concept. The Hyperledger Fabric is the framework underpinning the IBM blockchain platform.

The Hyperledger framework includes a distributed ledger, consensus algorithm, privacy, and smart contract. The distributed ledger in the Hyperledger Fabric can be separated into several channels based on the architecture developed. Hyperledger differs from the Ethereum blockchain, which consists of only one ledger for all transactions on the network, both for private and global Ethereum blockchains. The Hyperledger Fabric allows the creation of select channels that can be used by at least two nodes already connected to the system, or channels that consist of several nodes where the ledger is not incorporated into the global ledger. This channel is known as a private channel.

The difference between the Hyperledger framework and the Ethereum blockchain framework lies primarily in the permissioned blockchain. The Ethereum blockchain is based on global trust, where all nodes can freely join and interact with the Ethereum framework. Anyone can join the network. On the other hand, Hyperledger is a permissioned blockchain, where all participants joining the network know one another. Permission and verification are required as a form of consensus among all parties, proving who has already entered the Hyperledger network first.

Another fundamental difference is that Hyperledger does not require PoW to enter new data on the ledger. Consequently, the process of logging data into the Hyperledger's

blockchain is faster than the Ethereum blockchain's PoW concept. Every participant registered in the network is allowed to keep a copy of the system record. None of them, however, can change or delete the data. Therefore, Hyperledger still adopts the basic principles of a blockchain.

Based on the preceding description, it can be seen that the Hyperledger Fabric framework's fundamental differences reside in its architecture and topology, which are explained in detail in the next sections.

2. Network Topologies

The network setup used in this thesis consists of three organizations, i.e., a developer organization (ORG1), a client organization (ORG2), and a quality control organization (ORG3). Each organization has two peers at the beginning and up to six peers are added for scalability testing purposes, and each has a certificate authority (CA). In addition to the three organizations in the framework, the orderer cluster consists of three ordering service nodes that use a Kafka cluster composed of three zookeepers and four Kafka brokers. The network setup is illustrated in Figure 19.

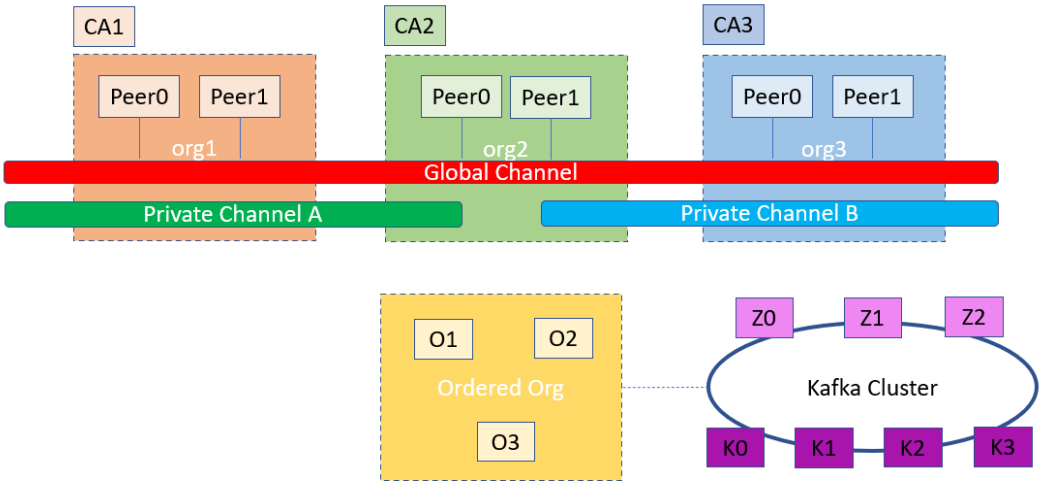

Figure 19. Hyperledger Fabric Node Topologies

In the network design shown in Figure 19, peers serve as the client of every organization. In the real world, peers are staff under an organization or sub-sections of the organization. For example, it can be assumed that peer0 is the Marines, and peer1 is an educational institution in the client organization. On the other hand, in the developer organization, peer0 may be considered the private developer. Each peer will copy every new ledger from the network.

CA on a Hyperledger Fabric network has two main components, i.e., CA server and CA client. Communication with the CA server can be done through the CA client connection or the fabric SDK connected directly to a peer. All communication through the CA client or SDK uses representational state transfer (REST) APIs. The order of authority hierarchies within the CA server follows the intermediate CA's hierarchies and the root CA. An intermediate CA has a parentROOT CA or another intermediate CA. Its main functions are registration of identities and enrollment certificate printing. A docker, i.e., a container engine that uses Linux kernel functions to run applications on an operating system, is required to run the CA server.

Another fundamental difference between the Hyperledger and Ethereum blockchains is on the orderer node. The orderer node architecture serves to eliminate vulnerability forks that exist on non-permissioned blockchains, where any transaction included in the block will not be verified as a final transaction. At the same time, the other peer(s) may be undertaking the fork process to verify another block. This condition differs significantly from the Hyperledger design, where the orderer node creates the ordering service, and each block is verified as a final block before proceeding to confirm another block.

In this Hyperledger design, a Kafka cluster is used to collect one or more servers adopting the API concept that handles the data record stream. The Kafka cluster consists of a zookeeper, producer, consumer, and brokers. The Kafka cluster has several brokers that function as a load balancer. Brokers carry out data streaming that can handle terabytes of messages. The zookeeper is responsible for managing and coordinating the work functions of the brokers. The producer functions to push data to brokers and obtain broker IDs, while the consumer functions to pull messages from brokers by getting an offset

update from the zookeeper. In general, Kafka is open-source software with a framework for storing and reading data streaming from the Hyperledger Fabric architecture.

3. Consensus

Consensus is the process by which a distributed ledger system reaches a mutual agreement among all nodes on a transaction or data value. The consensus design intends to develop a reliable transaction concept carried out by nodes that are not reliable. The consensus is a filter for the running of a system to avoid any possible threats. Consensus on the Hyperledger Fabric must ensure that each transaction block entered into the ledger has been validated previously. The consensus design must guarantee the correctness of the block to be entered into the ledger.

In this thesis, the Hyperledger Fabric adopts permissioned voting-based consensus. It is assumed that Hyperledger will be applied to networks with a moderate level of security, i.e., those with possible security gaps at an intermediate level. Thus, the algorithm with permissioned voting-based consensus is very suitable because it provides a process with low latency. Consensus on other blockchains requires most nodes to validate transactions or blocks, and thus the more nodes a network has, the more time is needed to reach consensus, which will affect scalability and speed. This consensus design also minimizes threats from unknown nodes because all members of the Hyperledger Fabric network are nodes that have been known and approved in advance.

D. COMPARISON OF APPROACHES TO ANALYSIS DESIGN

The calculation of the transaction time and scalability of the two frameworks using the ANUSD application is the core of this research. The expected outcome is reliable data that contains values of each experiment conducted on both frameworks. The burden given to both frameworks is a transaction in the form of history recording on the ledger. An experiment of 1–10,000 transactions is conducted on each framework to determine the transaction's average time and scalability and obtain a fixed value from this comparison. The appropriate performance analysis method is necessary to get valid research findings to generate accurate comparison results.

The performance analysis methods that can be applied to this calculation are manual methods and automatic methods. The first refers to methods that carry out the calculation of each process's transaction time and scalability using a manual timer. The entry of new transactions into the ledger indicates the start and end of the calculation. Time calculation begins immediately after the transaction's execution starts, i.e., when the transaction is entered into the ledger and each node receives the latest database update. It is also at that moment that calculation ends. The calculation can be performed using a stopwatch or timer on each node's computer. Due to a complex transaction processing mechanism on the blockchain and different mechanisms adopted by both frameworks, however, this research's expected goals, which are data accuracy, thoroughness, and validity, cannot be achieved with the application of a manual method. Another flaw of this method is human error, whose likelihood to occur is high when the transaction frequency exceeds the human ability to perform calculations. Manual methods can be performed only on a single transaction or multi-transactions that are regular and scheduled.

To avoid the flaws of manual methods in calculating each transaction, we need an automatic method. Both framework designs therefore adopt an automated method that uses Hyperledger Caliper. Hyperledger Caliper is "a blockchain benchmark tool, [and] it allows users to measure the performance of a blockchain implementation with a set of predefined use cases" [13].

The Hyperledger Caliper design applied in this thesis obtains input from each framework and then calculates each transaction's transaction time and scalability. As for the interface display, it uses a website as a visualization of each transaction. The architecture layer of the Hyperledger Caliper is illustrated in Figure 20.

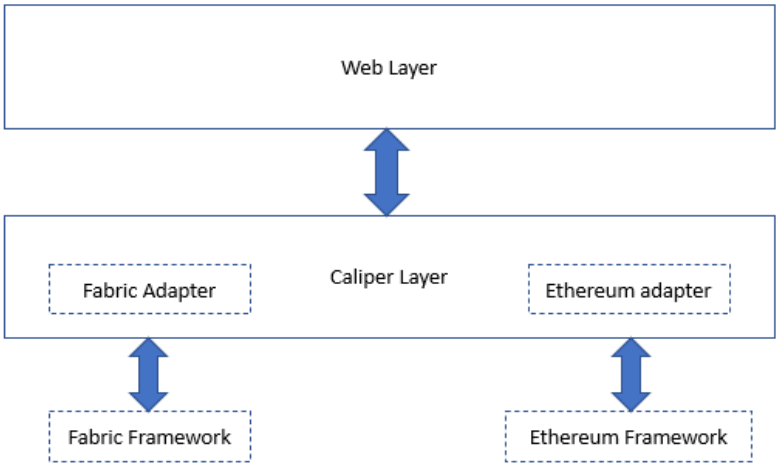

Figure 20. Hyperledger Caliper Layer Architecture

The caliper layer is the main component of the Hyperledger Caliper. This layer functions primarily to combine the implementation of the two frameworks to calculate block time and scalability. The web layer is a layer where this automatic method provides the visualization to users about block time and scalability calculation results. The display is presented in the form of data.

Based on this design, the next part is implementation. In the next chapter, we describe how the web-based ANUSD works and communicates with the framework using a smart contract. A 1–10,000 transaction test is applied for each framework to calculate the transaction time and scalability rate.

IV. IMPLEMENTATION

This chapter discusses the implementation of the design that has been outlined previously in Chapter III. The ANUSD application is implemented in the two frameworks. Hyperledger and Ethereum transaction testing is conducted using two methods, namely the determination of latency (transactions per time) and the scalability measurement methods. Latency measurements are carried out by varying the application size in every trial phase for each framework. As for the scalability test, several additional nodes are added to each organization to see the difference in latency in the process of forming new blocks in each framework.

A. AN APPLIED ANUSD ON A PRIVATE ETHEREUM BLOCKCHAIN NETWORK

This section provides step-by-step instructions for installing a local private Ethereum network, and testing the implementation of the ANUSD application to obtain accurate data on latency and scalability. A local private Ethereum network is built on an Ubuntu 18.04 LTS operating system and AWS cloud computer system.

To conduct latency (transactions per second) testing on the Ethereum network, the following scenario is conducted:

- The latency test is performed on three peers and one miner node by applying 10–10,000 software delivery transactions from each peer randomly, in which each peer represents the client, quality control, and developer organizations, as explained in Chapter III. This test is hereinafter referred to as Test A.

- The latency test is performed on three peers and two miner nodes. This test involves two miners to see what difference, if any, occurs in latency due to the addition of new miners. This test is hereinafter referred to as Test B.

- The latency test is performed on six peers and two miners. This test is intended to reveal any difference in latency with the addition of more peers and miners. This test is hereinafter referred to as Test C.

1. Required Software and Installation

Based on the scenario described earlier, the latency testing requires six peers and two miners in the Ethereum network installation process on the AWS. Each node built on the AWS has a different public IP address (Multinode). The technical specifications of each node are listed in Figure 21. For the list of nodes prepared on the AWS, see Figure 22.

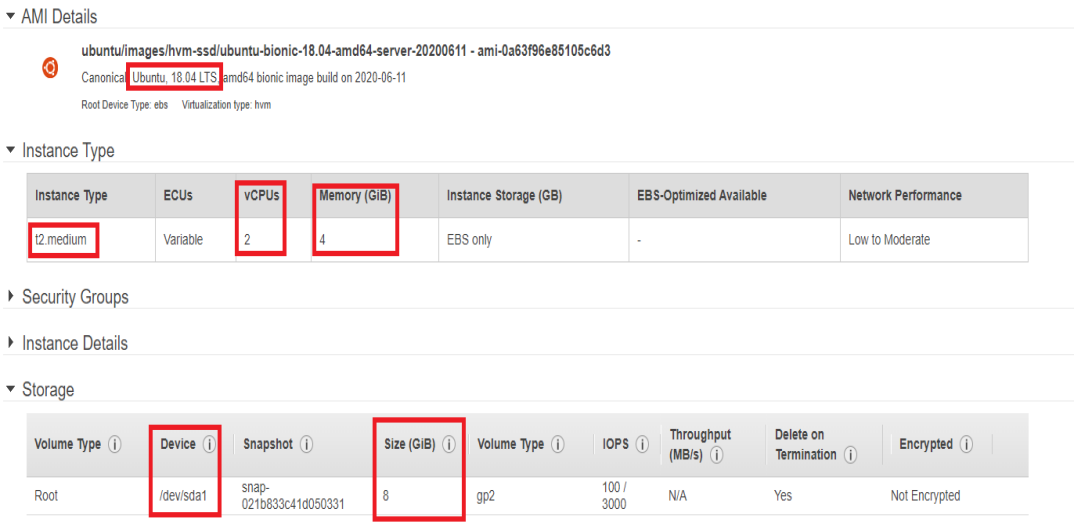

Figure 21. Technical Specifications of the AWS Peer Ethereum Node

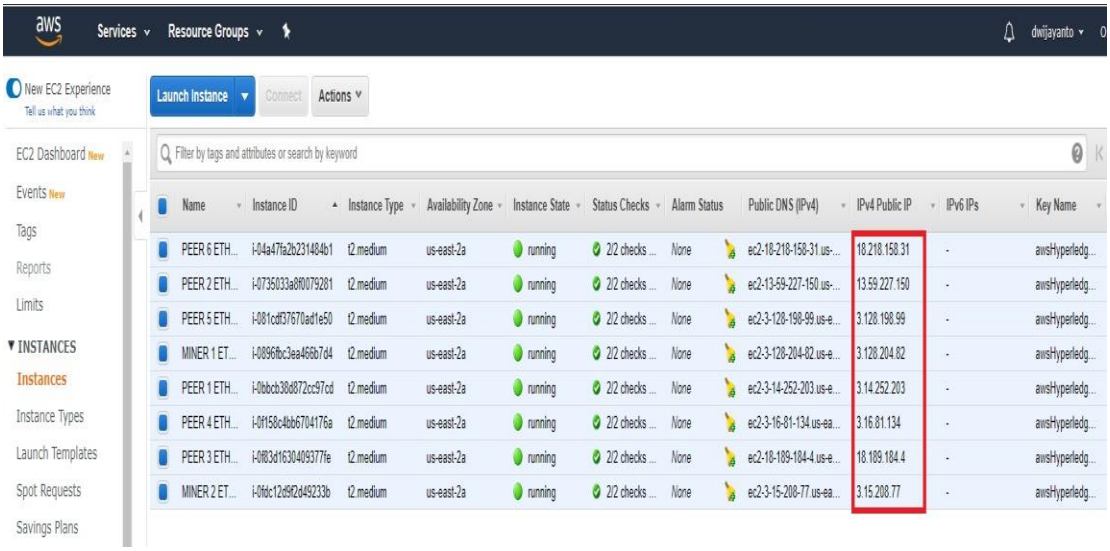

Figure 22. List of Nodes on the AWS

To start building an Ethereum private network, it is necessary to use some extra software to support the operation of this network. The first software needed is the Go programming language. Go language is an open-source tool designed for distributed systems and highly-scalable network servers. Go language is compiled and executed directly into machine language, which makes Go language faster compared to other programming languages that still need virtual machines. For our implementation, Go language version 1.9.3 is used.

After getting the source code from Go Ethereum and doing several basic housekeeping installation steps, the next step is to build the Ethereum network with the following command:

a. Cd~
b. Cd go-ethereum
c. Make geth

The next process to get this Ethereum private network running is to follow these steps:

a. Install BootNode. With Bootnode, each node on the Ethereum network will be easier to find by other nodes. Install Bootnode using this command: “sudo apt install bootnode.”

b. Make sure every node has Python 3 installed, using the command: “sudo apt install python3”

For convenience in the installation process, a package containing several installation folders for bootnode, miners, peers 1 to 6, and ANUSD smart contract test tools have been provided. With this package, it is expected that the installation process will be simpler and faster so that installation errors can be avoided. This package can be downloaded via the following link:

https://1drv.ms/u/s!AkWFvgCmANk761tbQYSykJdd3pf0?e=zlzwcc.

The list of installation files created to facilitate the Ethereum network installation process contained in the package file is shown in Figure 23, and one of the file examples created to facilitate the installation process is shown in Figure 24.

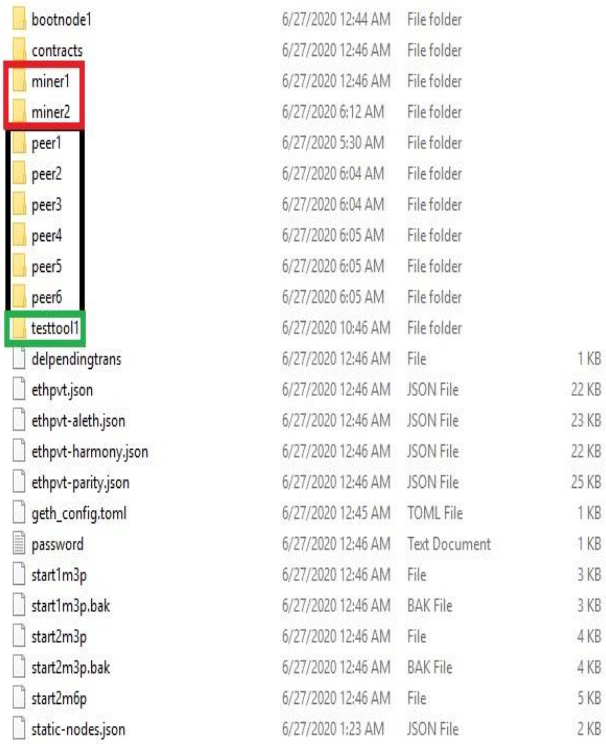

Figure 23. List of Installation Files in the Package

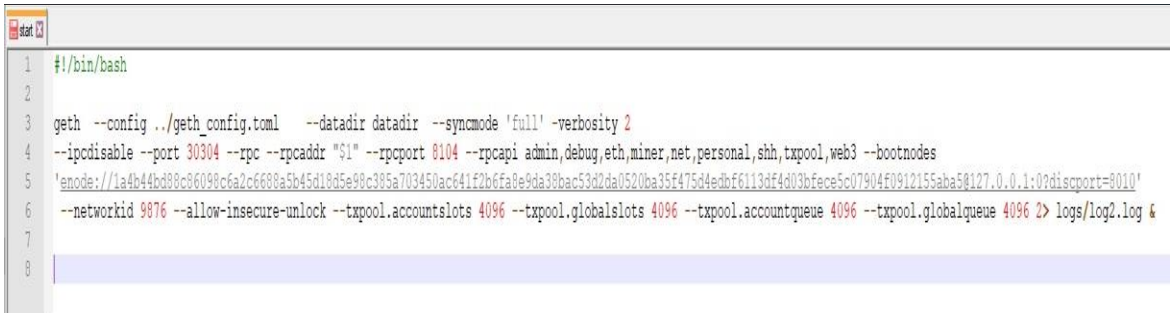

Figure 24. Example of a Start File to Run Peer1

The next step after downloading the installation package files is extracting the package file with the command: "*tar xvf dist1.tar.xz."* The extracted package file must be present in each peer (AWS peer). This can be done by copying all extracted files to each AWS peer using FTP.

In the next stage of the installation process for each peer in AWS, the installation folder that must be opened is adjusted to each peer type. For example, if the peer functions as miner 1, the installation starts by opening the Miner1 folder contained in the package file. This also applies to other peers, namely peer 1 to peer 6.

The next process to activate the Ethereum framework is to enable bootnode, which must be installed on only one peer. For example, for Test A, bootnode can be installed on peer 1, 2, or 3. Bootnode does not need to be installed on all peers. The bootnode installation process can easily be done using the command "*./start <ip address peer>."* Specifically, if bootnode is activated on peer 1, for example, then the command becomes "./start IPAddress-peers1." It must be ensured that the "*start*" file used is the file contained in the package folder in the intended peer. Hence, if you want to install bootnode on peer 1, then the "*start*" file used is contained in the folder of: "*../eth/bootnode1*" in the installation package file that has been downloaded on peer 1. After bootnode is activated on one of the peers, we can make sure the bootnode process runs by entering the command: *"ps -f."* If bootnode runs successfully, it will appear as shown in Figure 25.

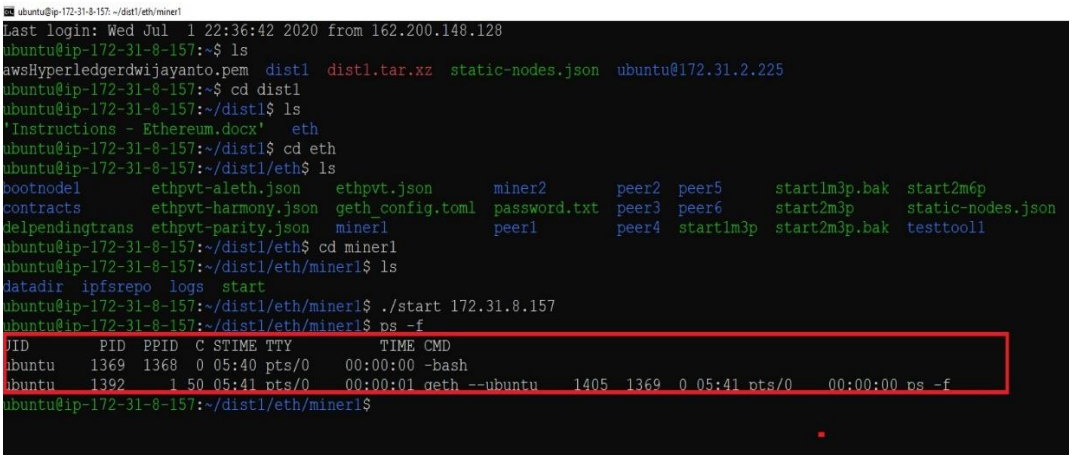

Figure 25. Bootnode Runs Successfully on Miner1

The next process is to activate miner 1 and peers 1 to 3 for the latency Test A, or activate miners 1 and 2 and peers 1 to 3 for the latency Test B, or activate miners 1 and 2 and peers 1 to 6 for the latency Test C scenario. The process of activating miners and peers is done only by entering them into a folder according to the functions of each peer. For example, if you want to activate peer1, then go to the package folder "*../eth/peer1"* and use the following command "./start IPAddress-peers1." This must be done for each peer involved in the latency testing scenario. If each peer and miner has been successfully activated, the display will resemble the one shown in Figure 26.

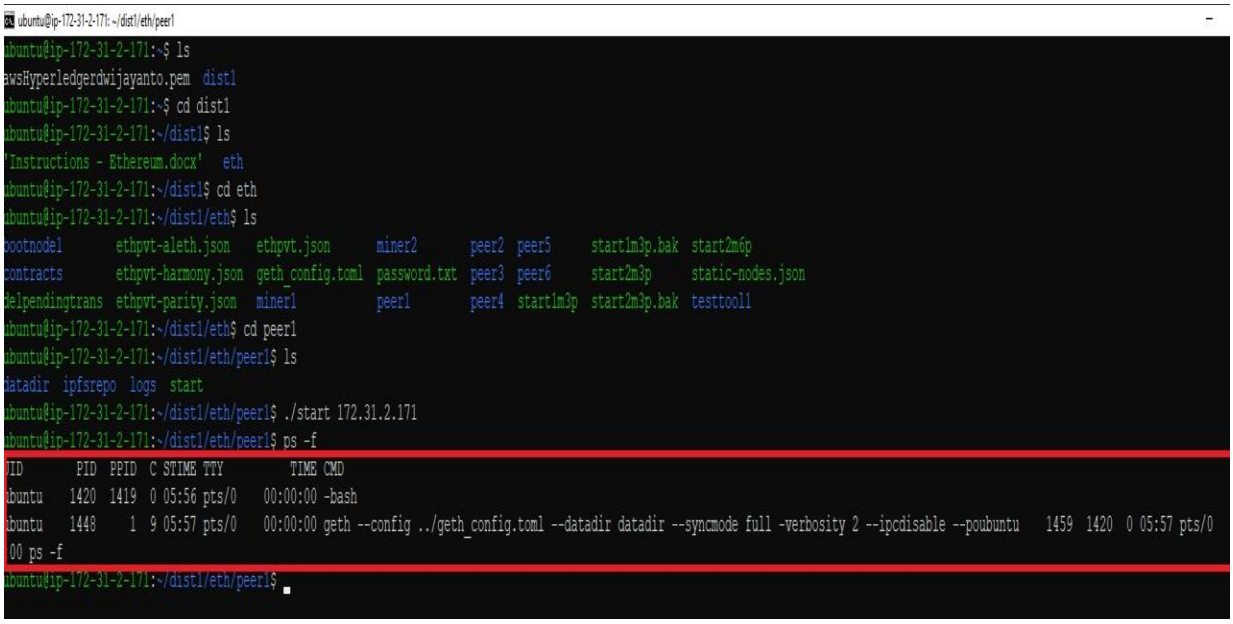

Figure 26. Peer1 Successfully Activated

After all miners and peers are activated, the next process is to launch the ANUSD smart contract that will support the latency testing scenario. The smart contract will function as an automatic console to carry out random transactions from one peer to another contained in the private Ethereum network installed previously.

2. Launching Smart Contract (D-apps)

The file size is not the most important part of this testing because the file size will not affect the latency (transactions per second) on the blockchain network. Blockchain is a system recording every transaction that has occurred before, as well as every change, either in the addition or reduction of transactions. Meanwhile, the file size will affect latency on off-chain networks (i.e., networks outside of the blockchain). More detailed logical steps of the data/software transfer process on the Ethereum network that supports ANUSD is explained in the following paragraphs:

- The client saves the file on the IPFS. According to a document about the InterPlanetary File System (IPFS) published by J. Benet [14], IPFS is a "peer-to-peer distributed file system that seeks to connect all computing

devices with the same system of files." In this testing scenario, the file transfer process will be handled in IPFS. Therefore, the resulting latency is the latency in the process of sending files from each peer to the IPFS system. Latency will depend on the type of equipment used by the peer and file size. It is not a latency factor, however, on the internal system blockchain. This is off-chain latency (outside the blockchain). Therefore, this testing does not consider the latency value. The off-chain latency value will vary greatly depending on the type of device used by the client/peer and the client's internet network towards the IPFS system.

- The next process involves the client retrieving the IPFS hash.
- Afterward, the client will submit a transaction to the blockchain with the IPFS file hash.
- Finally, the client will receive confirmation from the miner in the blockchain network. Thus, based on this logical step, the calculation of latency (transactions per second) will be calculated only in steps C and D. Steps C and D occur in the blockchain, and steps A and B are off-chain processes.

Smart contract in this testing has several main functions:

- Setting the maximum number of transactions on the network. This is adjusted to the trial scenario, which is divided into ten iterations, 100 iterations, 1,000 iterations, and 10,000 iterations. This means that the maximum number of transactions that will occur on the Ethereum network is 10, 100, 1,000, and 10,000 iterations, respectively, in each trial.
- Setting the distribution of the number of transactions in each maximum total iteration. For example, in a trial session of a maximum of 100 iterations, the smart contract will divide automatically and randomly several transactions carried out on peers 1, 2, and 3. For example, peer 1

will perform 33 transactions, peer 2 will perform 35 transactions, and peer 3 will perform 100- (35 +33) transactions.

- Performing the process of submitting transactions to the Ethereum blockchain network automatically and calculating the average transactions per second.

Running a smart contract can be done from one peer that is not incorporated in the testing scenario. For example, in Test A, which consists of three peers and one miner, the smart contract should be run on peer 4 so that it utilizes peer 4, which is not included in the testing scenario. This is to get accurate TPS calculation results because the peers involved in the scenario have their own transaction burdens that certainly affect the time of the TPS.

A smart contract can be run with the following command.

- Test A (three peers and one miner):

  1) cd ../eth/testtool1
  2) ./bctest.py -p "<IP of peer1>:8104" "<IP of peer2>:8105" "<IP or peer3>:8106" -i <iteration>
  The number of iteration trials can be adjusted to 10, 100, 1,000, or 10,000.

- Test B (three peers and two miners) orders are the same as in Test A and must be carried out on other peers that are not included in the testing scenario.

- Test C (six peers and two miners) commands are the same as those in Test A or Test B.

As an example, Figure 27 illustrates the process of preparing the Test A scenario, in which the smart contract testing will involve three peers, one miner, and one peer test tool. Connection with AWS is done using Secure Shell (SSH) with pre-determined credentials and private keys.

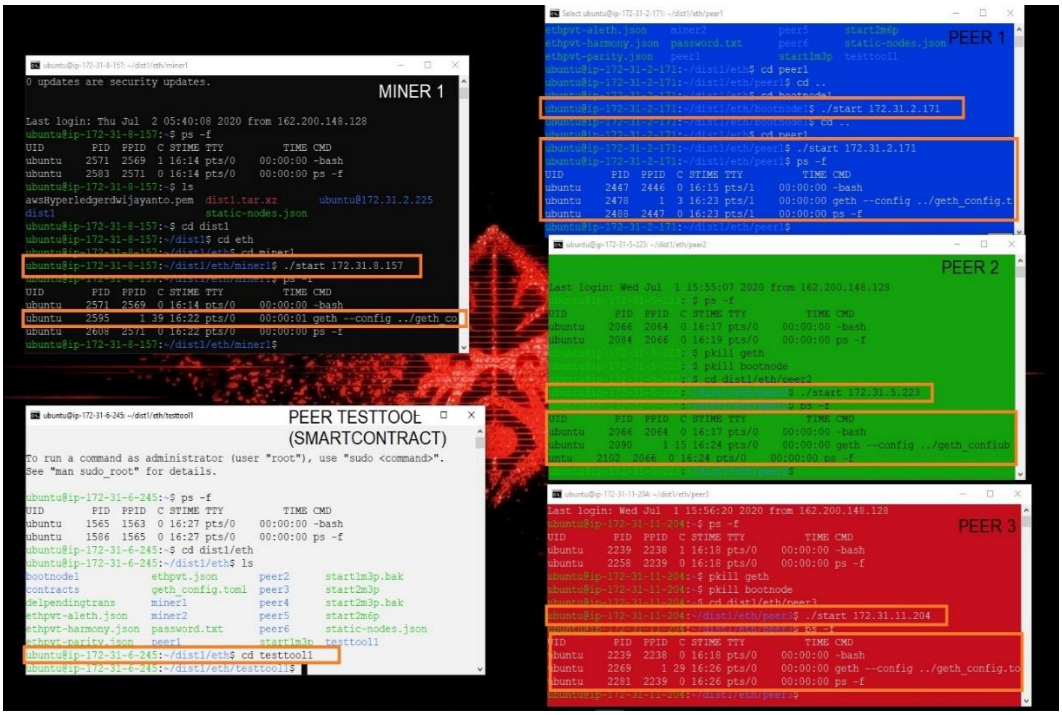

Figure 27. Example of Preparing the Smart Contract Activation for Test A

3. Testing the System

In the latency (TPS) testing process as previously explained, the testing is divided into three scenarios. In Test A, which includes three peers and one miner, a test is conducted to see the average transactions per second that Ethereum can handle with the implementation of the ANUSD architecture, which consists of one client peer, one distributor peer, and one quality control peer. The first process is to activate the test tool. The test tool will use the bctest.py file contained in the installation package file. In the first test, ten iterations will be carried out, so the command used on the peer test tool is:

"./bctest.py -p 172.31.2.171:8104 172.31.5.223:8105 172.31.11.204:8106 -i 10"

In the first test with ten iterations, some of the nodes used are node 1, peers 1 to 3, and peer 6. Peer 6 functions as a peer test tool. Some of the nodes activated in the first test are shown in Figure 28.

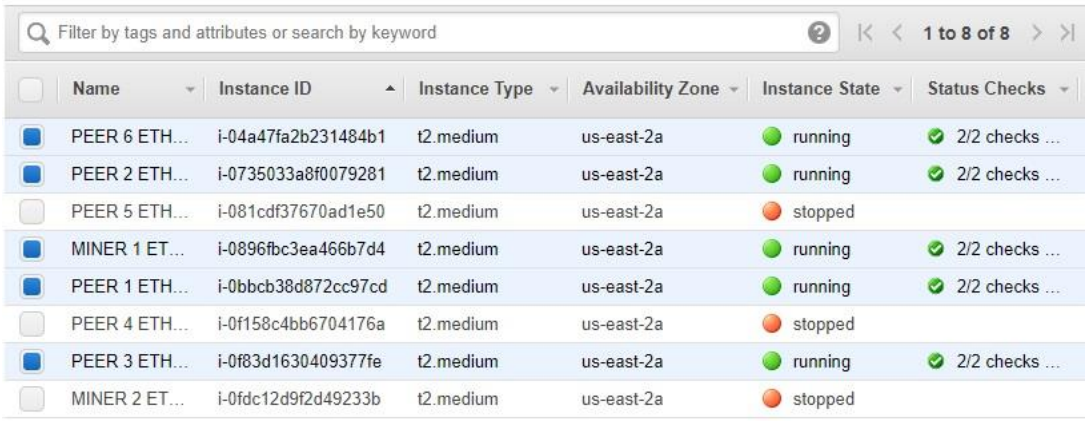

Figure 28. Activated Nodes for the First Test Run

After all nodes are active, the next process is to run miner 1 and peers 1 to 3 with the commands that have been outlined in the previous section. Meanwhile, only peer 6 runs the bctest.py command. The results of this ten-iteration testing are shown in Figure 29.

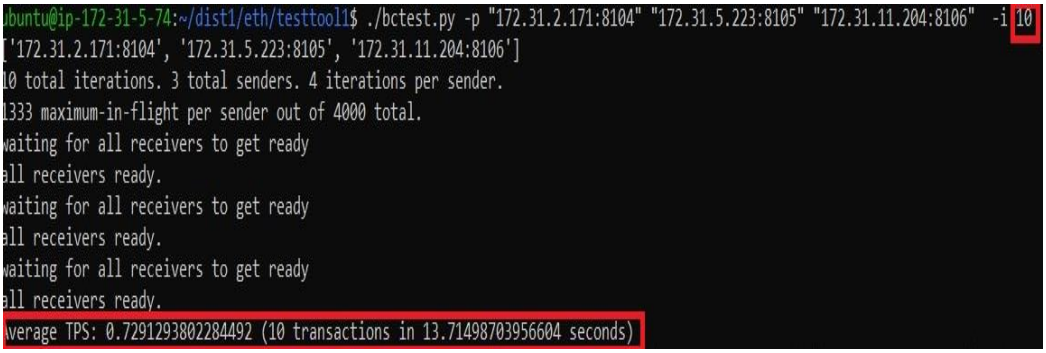

Figure 29. Test A: Results of 10 Iterations

After the ten iterations are successful, 100 iterations are performed using the following command:

"./bctest.py -p 172.31.2.171:8104 172.31.5.223:8105 172.31.11.204:8106 -i 100"

The results of testing with 100 iterations are shown in Figure 30.

.

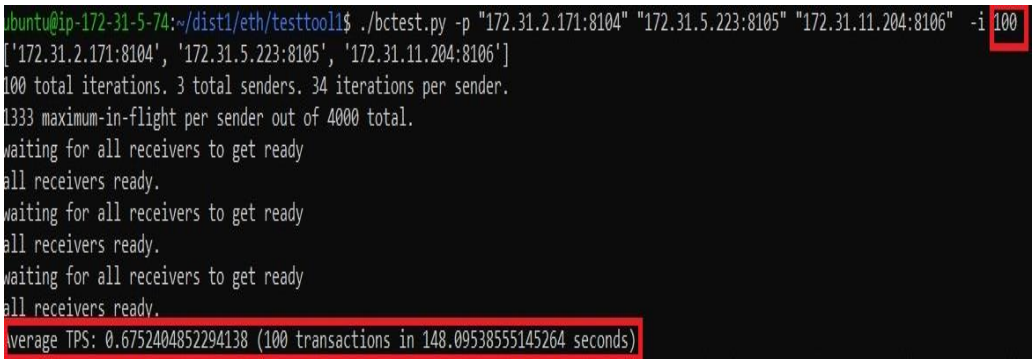

Figure 30. Test A: Results of 100 Iterations

The next processes are 1,000 iterations and 10,000 iterations. For each, the command used will be slightly different. The difference lies in the number of iterations to be performed. There is a little change at the end of the bctest.py command. The results of 1,000 and 10,000 iterations can be seen in Figure 31 and Figure 32, respectively.

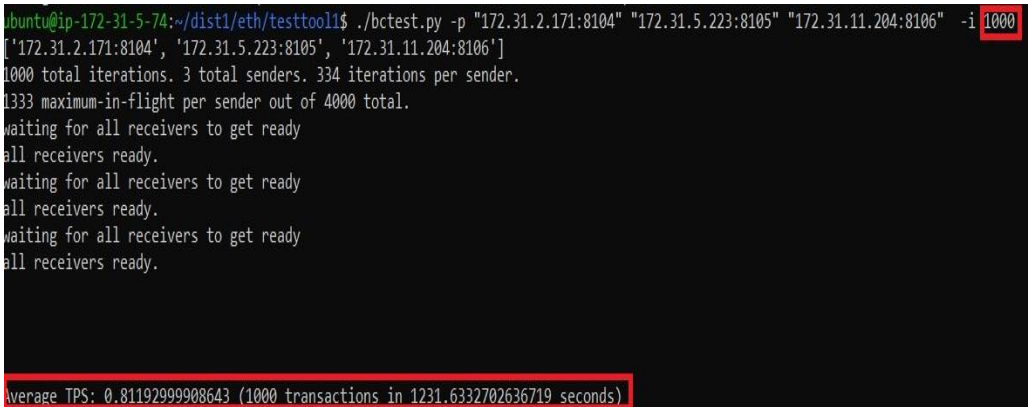

Figure 31. Test A: Results of 1,000 Iterations

```
buntu@ip-172-31-15-16:~/dist1/eth/testtool1$ ./bctest.py -p 172.31.2.171:8104 172.31.5.223:8105 172.31.11.204:8106 -i 10000
'172.31.2.171:8104', '172.31.5.223:8105', '172.31.11.204:8106']
as Limit: -1
0000 total iterations. 3 total senders. 3334 iterations per sender.
333 maximum-in-flight per sender out of 4000 total.
aiting for all receivers to get ready
ll receivers ready.
aiting for all receivers to get ready
ll receivers ready.
aiting for all receivers to get ready
ll receivers ready.

verage TPS: 0.57457296965207 (10000 transactions in 17404.229798793793 seconds)
```

Figure 32. Test A: Results of 10,000 Iterations

The completion of 10,000 test iterations marks the completion of the first test with three peers and one miner. We can now make a TPS chart for Test A, as shown in Figure 33.

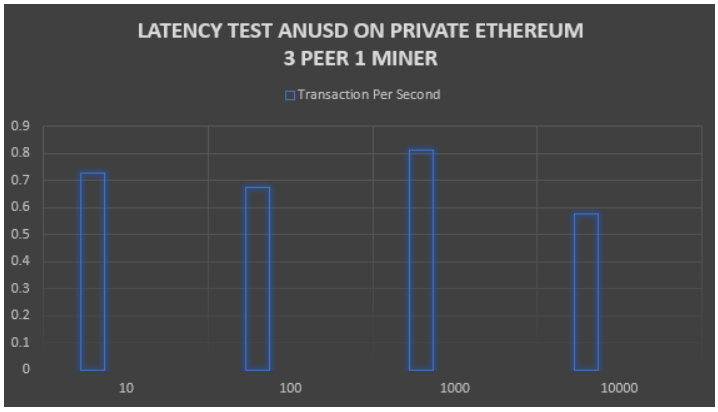

Figure 33. TPS Chart for Test A (Three Peers and One Miner)

It is important to note if an error occurs during the testing process, the network connections must be checked before running a smart contract. Several actions can be taken to check whether the network is installed correctly:

a. Connect to a peer through the "geth console," e.g., let's check peer3

   geth attach https://<peer3's IP>:8106

b. Once connected, confirm that it is listening by running:

> **net.listening**

It should print:

true

c. Check connectivity to all three other nodes, as in the first test:

> **net.peerCount**

It should print:

3

For Test B (three peers and two miners) the sequence of installation steps differs only slightly from Test A. The server that needs to be activated is miner 2, and the bctest.py command is used as it was in Test A. The results of Test B can be seen in Figures 34, 35, 36, and 37.

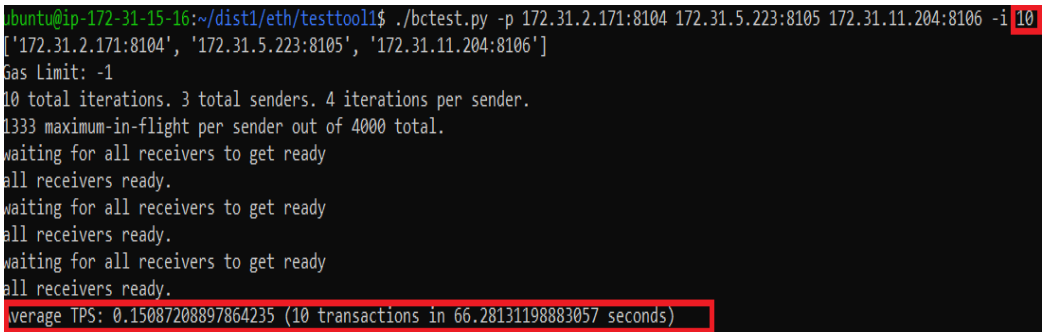

Figure 34. Test B: Results of 10 Iterations

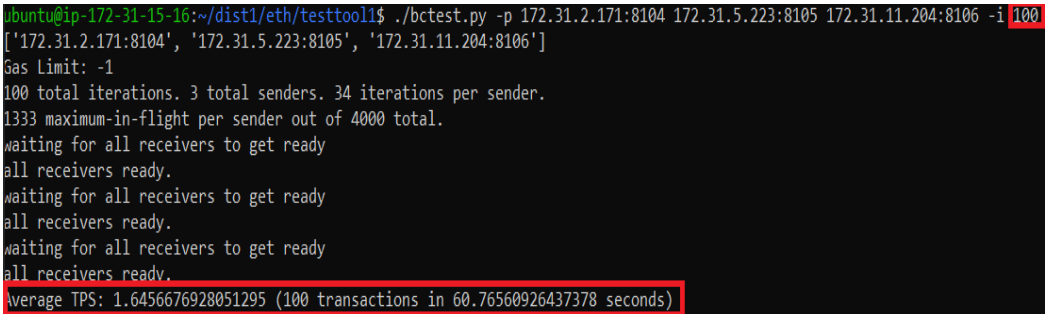

Figure 35. Test B: Results of 100 Iterations

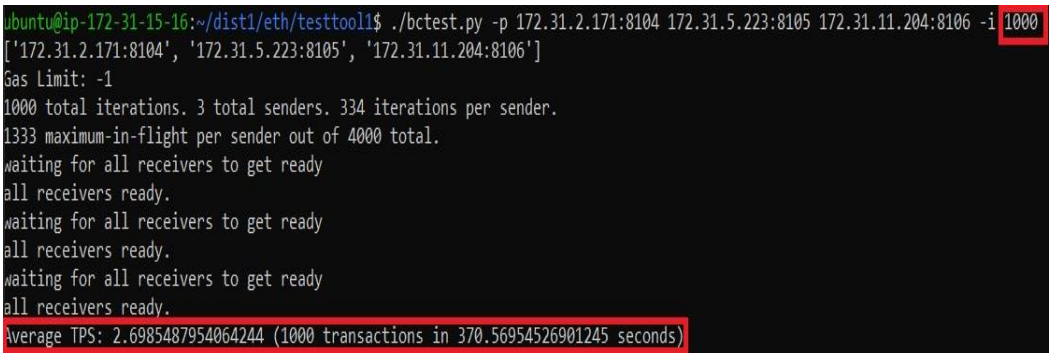

Figure 36. Test B: Results of 1,000 Iterations

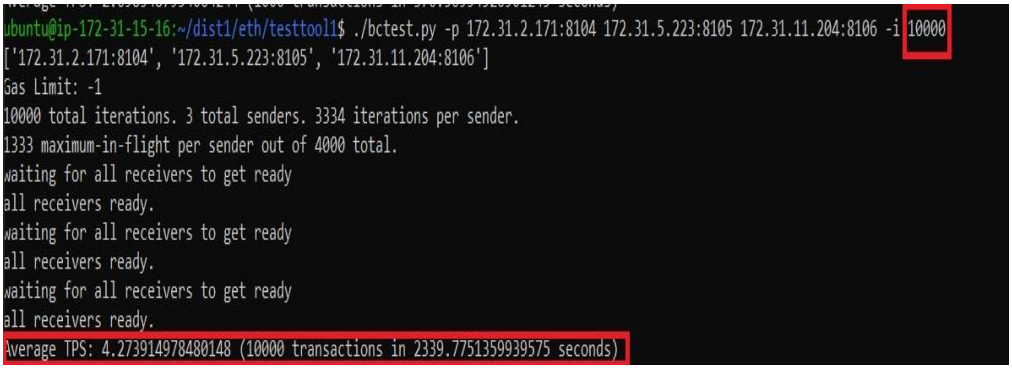

Figure 37. Test B: Results of 10,000 Iterations

The completion of 10,000 test iterations marks the completion of Test B with three peers and two miners. We can now make a TPS calculation for Test B, as shown in Figure 38.

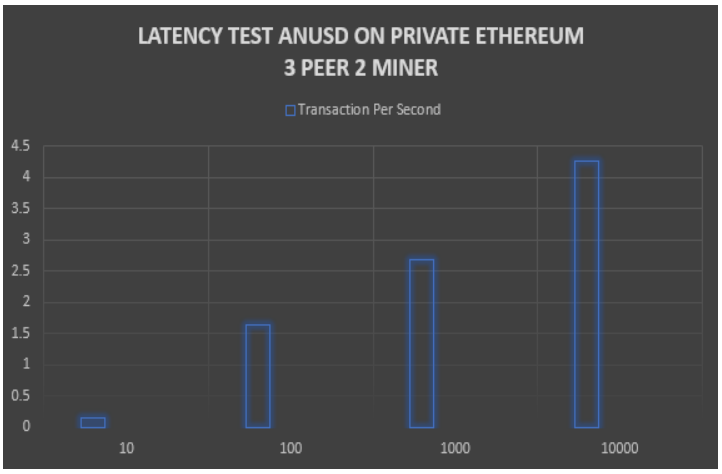

Figure 38. TPS Chart for Test B (Three Peers and Two Miners)

The next test, Test C, consists of six peers and two miners. To perform this test, it is necessary to activate peers 1 to 6 and miners 1 and 2. The command used to activate the peers and miners is the same as the command described for the previous two tests. One difference in this testing with six peers is the need to add one new peer in the AWS for the bootnode and test tool. Test tools and bootnode must be performed for other peers outside the core peers included in the scenario. This is done merely to get a more accurate TPS calculation without the burden of interference from other peers. Because the core peers involved in the scenario have their respective burdens on each transaction, it is expected that the burden of benchmarking transactions per second will not increase the value of TPS. The difference in the commands for in Test C is only in the test tool command. It is the addition of IP on peers 4 to 6. The following is the adjusted command for the test tool:

./bctest.py -p "<IP of peer1>:8104" "<IP of peer2>:8105" "<IP or peer3>:8106" "<IP or peer4>:8106" ***"<IP or peer4>:8108" "<IP or peer5>:8109" "<IP or peer6>:8110"*** *-i 100*

The results for Test C (six peers and two miners) can be seen in Figures 39, 40, 41, and 42.

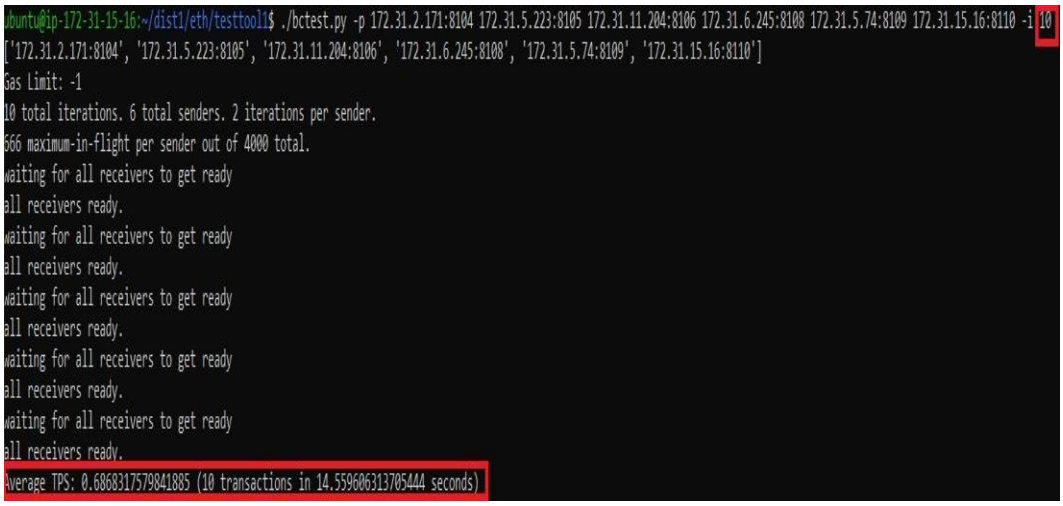

Figure 39. Test C: Results of 10 Iterations

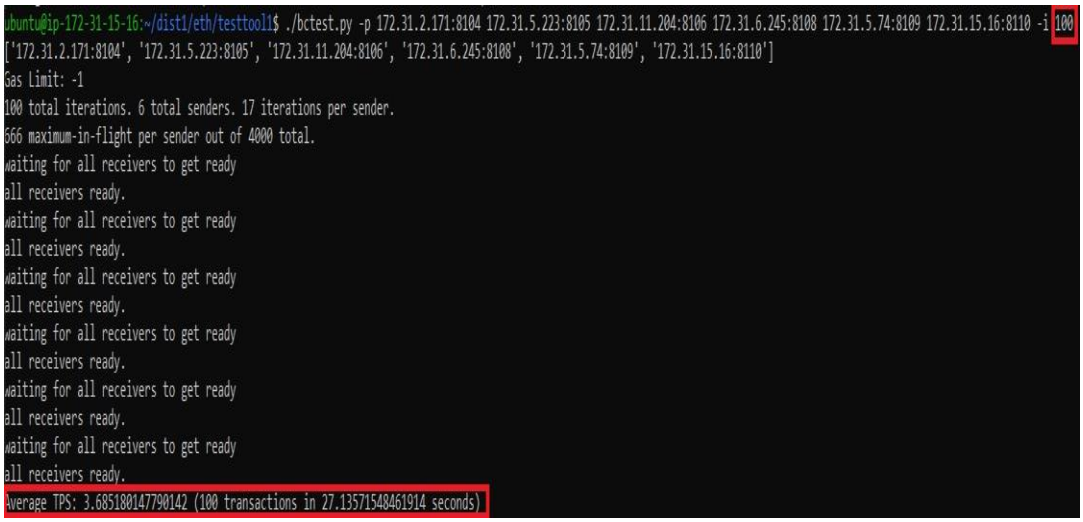

Figure 40. Test C: Results of 100 Iterations

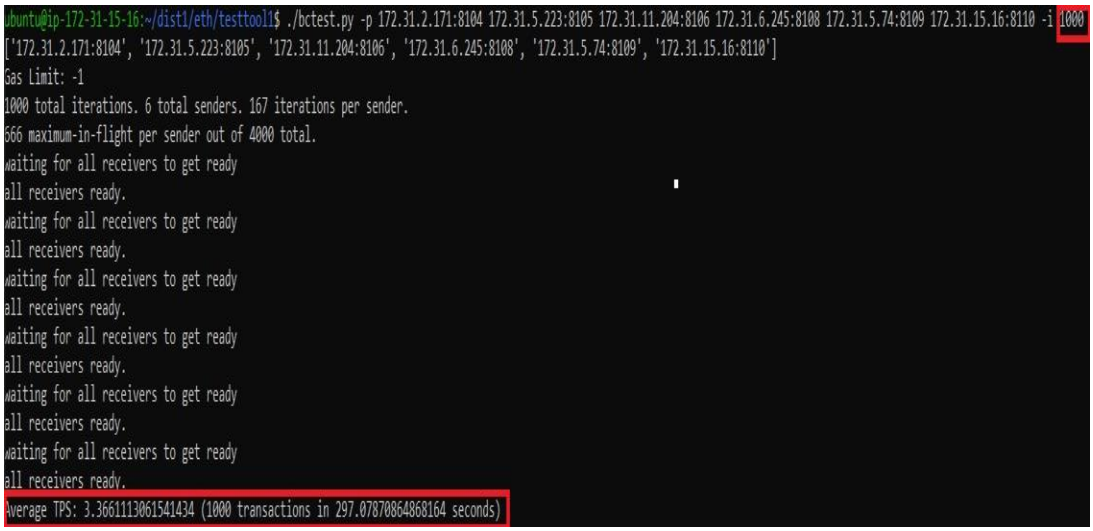

Figure 41. Test C: Results of 1,000 Iterations

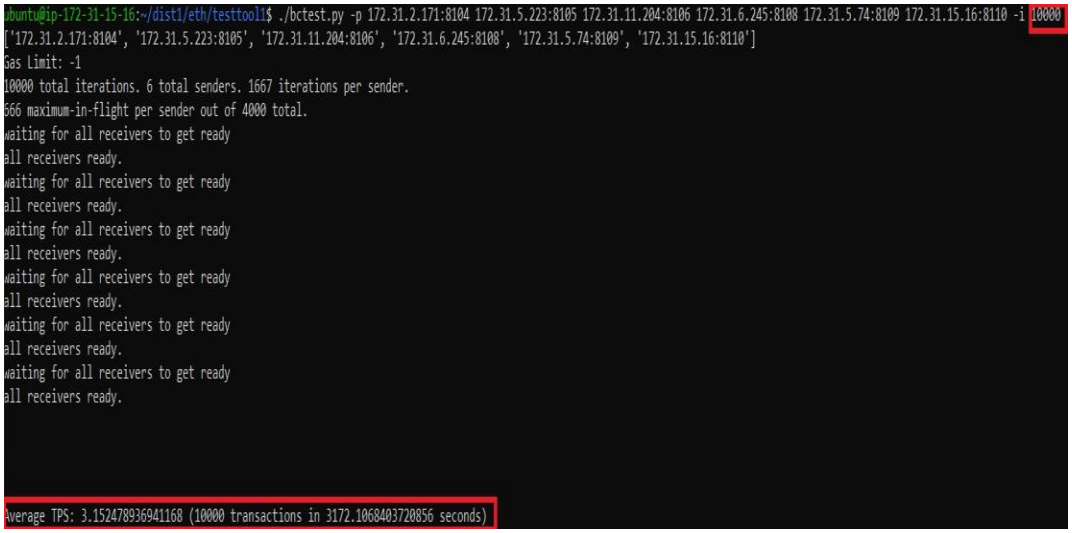

Figure 42. Test C: Results of 10,000 Iterations

The completion of 10,000 test iterations marks the completion of Test C with six peers and two miners. We can now make a TPS calculation for Test C, as shown in Figure 43.

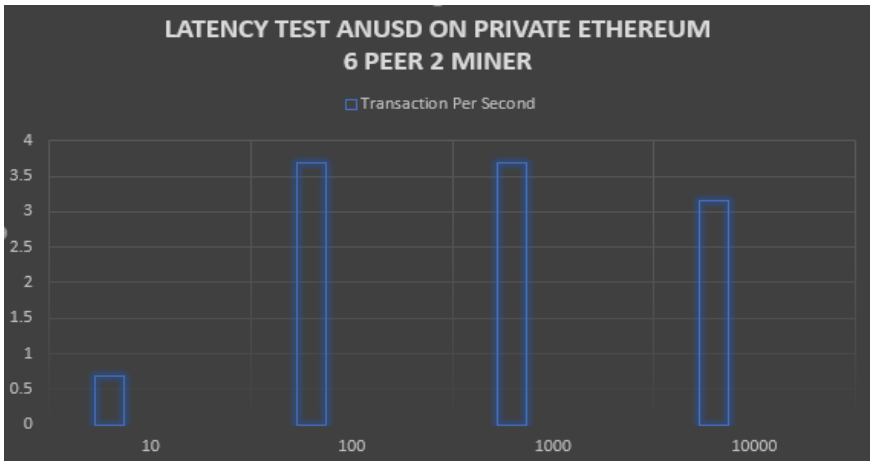

Figure 43. TPS Chart for Test C (Six Peers and Two Miners)

With the end of Test C, all testing to calculate the average transactions per second has been completed. The results of the analysis are explained in the next section. Some additional information that is worth knowing regarding this experiment is as follows:

- The network already has a smart contract deployed. The contract source code can be found for reference in:
  - eth/contracts/source
- The network is configured so that all nodes can be run in a single box (with a single IP) if required. This is achieved by assigning unique ports to each process. The unique port is shown in Table 1.

Table 1. List of Used Ports

Process	Ethereum Port	RPC API Port
miner1	30303	8103
peer1	30304	8104
peer2	30305	8105
peer3	30306	8106
miner2	30307	8107

Process	Ethereum Port	RPC API Port
peer4	30308	8108
peer5	30309	8109
peer6	30310	8110
bootnode1	8010	N/A

4. Analysis of Latency and Scalability

From the three test scenarios using the basic ANUSD architecture, it was possible to calculate the average transactions per second or latency and scalability and display the calculated TPS for each test in the charts display in Figures 33, 38, and 43, respectively. In this paper, we also provide an analytical summary of those results, which is shown in Figure 44.

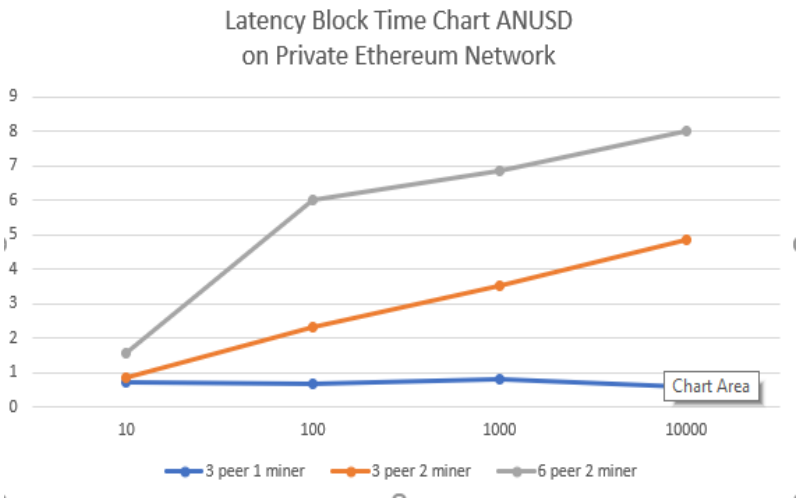

Figure 44. Final Latency TPS for ANUSD on Private Ethereum Network

B. AN APPLIED ANUSD ON A PRIVATE HYPERLEDGER NETWORK

This section provides step-by-step instructions for the installation of a local private Hyperledger network as a form of implementation of this research. To test the implementation of the ANUSD application to obtain accurate data regarding latency and scalability, the Hyperledger of the local private network is built on an Ubuntu 18.04 LTS OS and the AWS computer cloud system with a T2. A medium instance is applied to the private Ethereum network implementation in the previous subchapter. In this case, however, the virtual disk size used is larger. The reason is that the system orderer on the Hyperledger network requires more hard disk resources than Ethereum.

Latency (TPS) testing on the Hyperledger network is performed as described in the following scenarios:

- Latency testing is conducted on three organizations and two peers in each organization by applying 10–10,000 software delivery transactions from each peer randomly; each organization represents a client, quality control, and developer, as explained in Chapter III. This test is hereinafter referred to as Test A.

- Latency testing is conducted on three organizations and four peers in each organization; the addition of two more peers in each organization in this test is aimed to see what impact, if any, their addition has on latency. This test is hereinafter referred to as Test B.

- Latency testing is conducted on three organizations and six peers, similar to Test B, to see the effects of scalability on latency. This test is hereinafter referred to as Test C.

1. Required Software and Installation

Based on the scenarios described previously, the process of installing a Hyperledger network on the AWS requires a maximum of eight peers, because the total number of peers that will be used in Test C is six peers plus one order peer and one test node. Each node is

built on the AWS, which has a different public IP address (Multinode). The technical specifications of each node and the list of nodes prepared on the AWS can be seen in Figure 45.

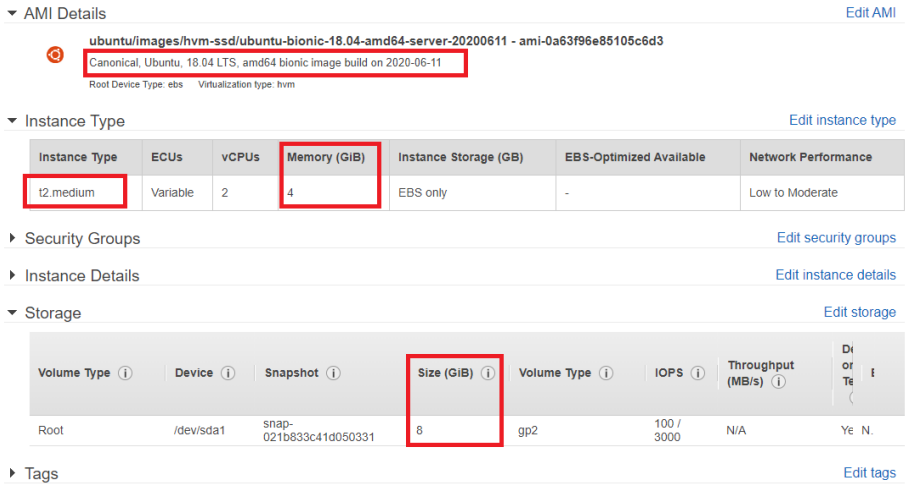

Figure 45. Technical Specifications of the AWS Node for Each Hyperledger Peer

The distribution process of the Virtual Machine (instance) in AWS is shown in the Figure 46.

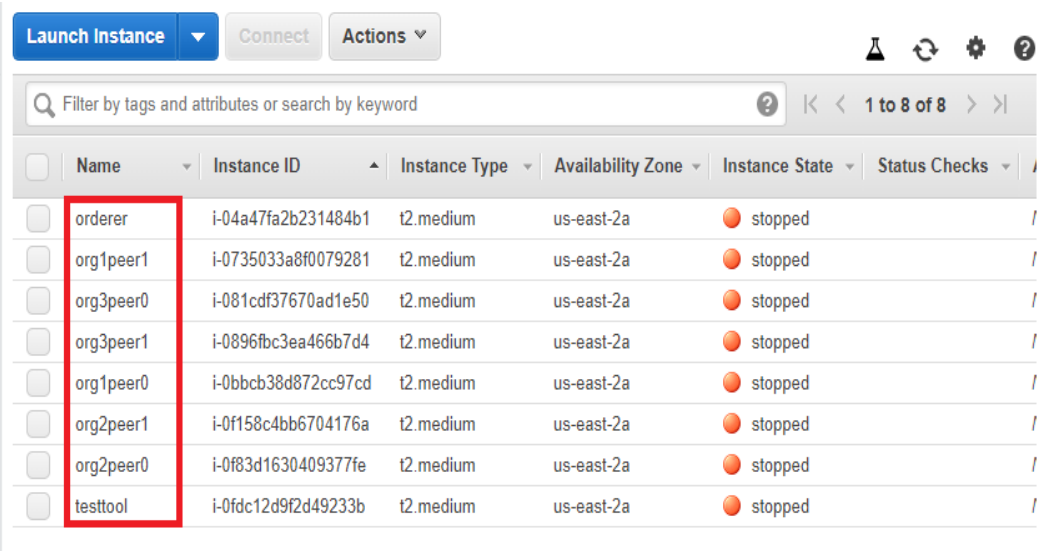

Figure 46. List of Nodes in the AWS for the Hyperledger Network

To begin installing the Hyperledger Fabric network, follow steps:

- Allocate eight boxes (real or VMs) and install the latest Ubuntu 18.04 LTS on it.
- Ensure that each box has at least 10 GB free space on the main (working) partition.
- Set up each box in the cluster to achieve the process distribution illustrated in the ‘Process Distribution’ section.
- Prepare each machine as described here: https://Hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html
- Install the samples and binaries for HLF **v1.4.7** on each box as described on the following page.
- After installing, check each box for successful installation by running the BTFN sample. Build Your First Network (BYFN) and see if the process runs to completion.

    1) $ cd fabric-samples/first-network

    2) $ cd fabric-samples/first-network

    3) $ ./byfn.sh up

And bring down the BYFN with

    4) $ ./byfn.sh down

- Make sure that each box is in the cluster of the necessary Nodes setup and tuned in the folder: ~/hlf147/fabric-samples/ut/
- Copy that folder very carefully to the folder of each respective new box

- Check inside the folder titled ‘deployment’ (in ~/hlf147/fabric-samples/ut/deployment), which contains docker-compose scripts for starting each node—e.g., In the VM1 box:

- ubuntu@Org1Peer0-ip-172-31-2-171:~/hlf147/fabric-samples/ut/deployment$la-l <== VM1 box, inside the deployment folder

1. In each box, for each test, start ONLY the relevant processes by changing to the deployment folder via the provided docker-compose script:

   1) To bring up nodes for Test A, run:
   ~/hlf147/fabric-samples/ut/deployment$ docker-compose -f docker-compose-org1peer0.yml up -d
   2) To bring up nodes for Test B, run:
   ~/hlf147/fabric-samples/ut/deployment $ docker-compose -f docker-compose-org1peer0.yml -f docker-compose-org1peer2.yml up -d \
   3) To bring up nodes for Test C, run:
   ~/hlf147/fabric-samples/ut/deployment $ docker-compose -f docker-compose-org1peer0.yml -f docker-compose-org1peer2.yml -f docker-compose-org1peer4.yml up -d

- After running docker-compose, check whether the intended containers are running by doing a docker ps.

- Each of the previous docker-compose*.yml files was specifically tailored for the instance and the VM on which it is running. To customize nodes, we need to adjust the following Docker recommendations and guidelines:

  1) Follow the Docker Compose file from docker.com
  2) A simpler tutorial on setting up instances for HLF via Docker Compose can be found at: https://medium.com/@kctheservant/demo-of-three-node-two-channel-setup-in-Hyperledger-fabric-54ba8a9c461f

- Do the same in all boxes VM1–VM6.

- In VM8, similar to previously, start the orderer – this step is the same for each of the tests:

~/hlf147/fabric-samples/ut/deployment $ docker-compose -f docker-compose-orderer.yml up -d

- n VM9, bring up the API CLI by:

~/hlf147/fabric-samples/ut/deployment $ docker-compose -f pcli.yml up -d

2. Launching Smart Contract (D-apps)

The Hyperledger Fabric supports smart contract development in multiple languages. Official support is provided for a few languages such as Java, Node (JavaScript), and Go. The software development smart contract for this test was developed using the Java SDK already installed on the AWS cluster. The source code for the smart contact can be downloaded from the following link: https://1drv.ms/u/s!AkWFvgCmANk77CctLlbww2PIKMle?e=tdf5Ok

To running smart contract to test latency (TPS) on the Hyperledger network, use the following commands:

a. Test A

10–10,000 iteration tests:
ubuntu@TTip-172-31-2-225:~/hlf147/testtool3$ ./hlftest.py -tn a -I 10–10000

b. Test B

10–10,000 iteration tests:
ubuntu@TTip-172-31-2-225:~/hlf147/testtool3$ ./hlftest.py -tn b -I 10–10000

c. Test C

10–10,000 iteration tests:

ubuntu@TTip-172-31-2-225:~/hlf147/testtool3$ ./hlftest.py -tn c -I 10–10000

3. Testing the System

In the latency (TPS) testing process as previously explained, the testing is divided into three scenarios. Test A consists of two peers in each organization. Test B consists of four peers in each organization, and Test C has six peers in each organization. Each scenario is charged a transaction of 10–10,000 iterations. The testing is carried out to see the average transactions per second that can be done by the Hyperledger Fabric by applying the ANUSD architecture.

The testing results from the three tests can be seen in Figures 47 through 59.

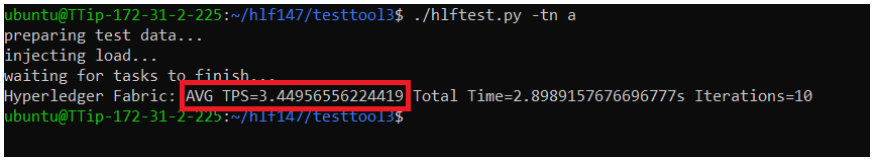

Figure 47. Test A: TPS for 10 Iterations/Transactions (Two Peers in Each Organization)

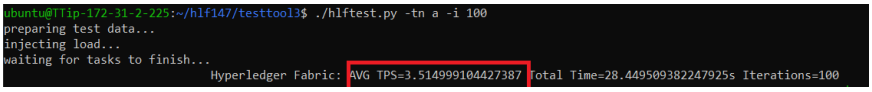

Figure 48. Test A: TPS for 100 Iterations/Transactions (Two Peers in Each Organization)

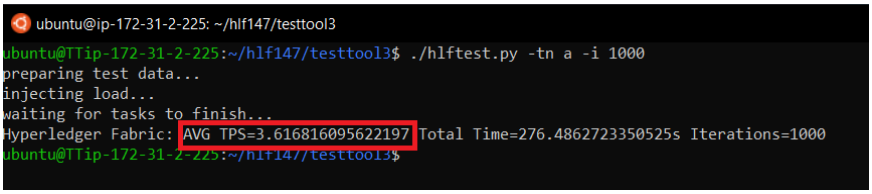

Figure 49. Test A: TPS for 1,000 Iterations/Transactions (Two Peers in Each Organization)

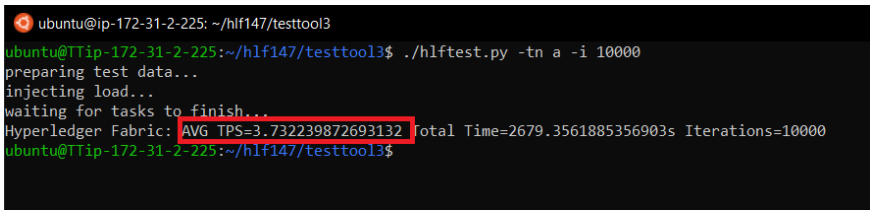

Figure 50. Test A: TPS for 10,000 Iterations/Transactions (Two Peers in Each Organization)

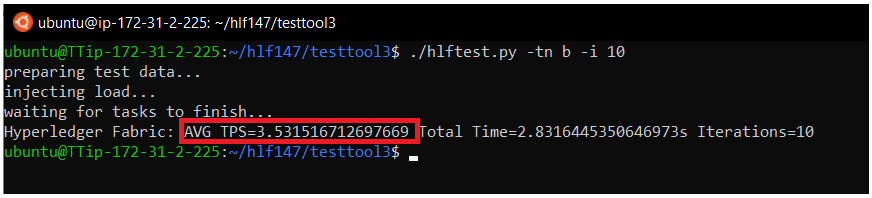

Figure 51. Test B: TPS for 10 Iterations/Transactions (Four Peers in Each Organization)

Figure 52. Test B: TPS for 100 Iterations/Transactions (Four Peers in Each Organization)

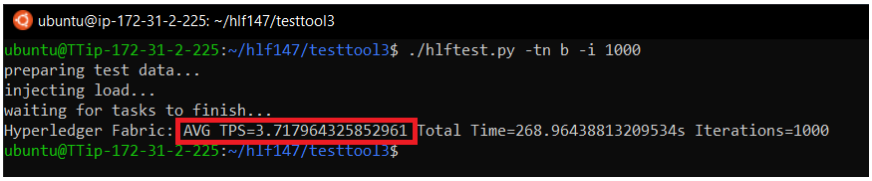

Figure 53. Test B: TPS for 1,000 Iterations/Transactions (Four Peers in Each Organization)

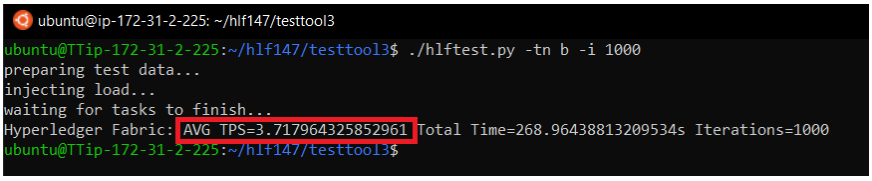

Figure 54. Test B: TPS for 1,000 Iterations/Transactions (Four Peers in Each Organization)

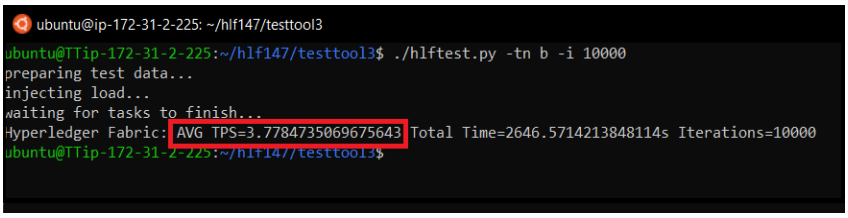

Figure 55. Test B: TPS for 10,000 Iterations/Transactions (Four Peers in Each Organization)

Figure 56. Test C: TPS for 10 Iterations/Transactions (Six Peers in Each Organization)

Figure 57. Test C: TPS for 100 Iterations/Transactions (Six Peers in Each Organization)

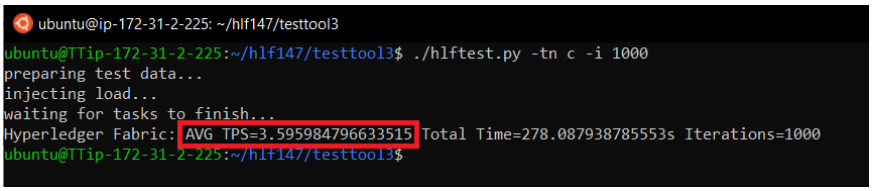

Figure 58. Test C: TPS for 1,000 Iterations/Transactions (Six Peers in Each Organization)

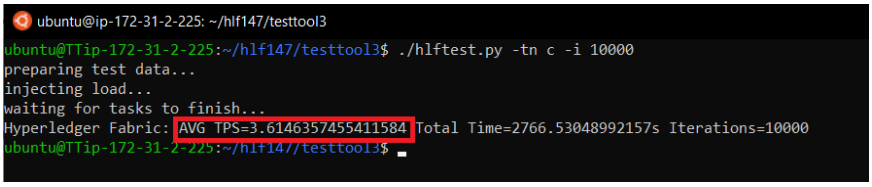

Figure 59. Test C: TPS for 10,000 Iterations/Transactions (Six Peers in Each Organization)

4. Analysis of Latency and Scalability

From the three tests using the basic ANUSD architecture, it was possible to calculate average transactions per second or latency and scalability. We provide a visual analytical summary of the three test scenarios in Figure 60. The chart shows that the addition of peers in each Hyperledger organization does not affect the calculation of the TPS value. The average number of transactions per second is between 3.4 and 3.7 TPS. Therefore, it can be concluded that the addition of peers on the Hyperledger network with the ANUSD application does not have a significant impact on the value of latency (TPS). This can serve as a reference for U.S. Navy decision makers in their selection of the most suitable blockchain technology, based on latency and scalability.

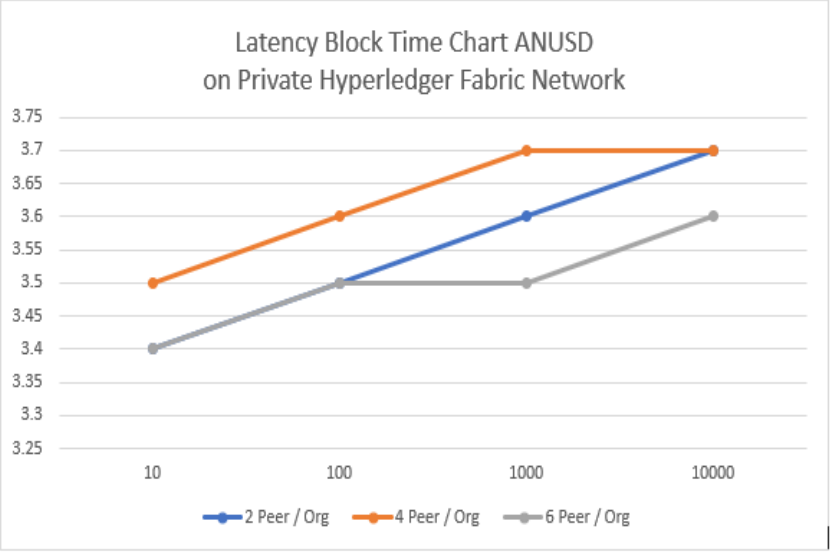

Figure 60. Comparison Summary of Latency (TPS) for ANUSD on a Private Hyperledger Fabric Network

V. CONCLUSION AND FUTURE WORK

This chapter concludes the thesis and offers recommendations for future work. The conclusion formulated in this chapter is based on the results of the experiments on the implementation of the Autonomous Navy Unclassified Software Distribution (ANUSD) in the Ethereum and Hyperledger private networks. The conclusions are also based on the chart summarizing latency (transactions per second) obtained from the tests conducted in Figure 60 at the end of Chapter IV, and what was observed about the effect of scalability on latency. This chapter also discusses the obstacles encountered during the experiment. Further, the chapter recommends possible areas to explore in future work that can be continued by other researchers, who can expand on topics beyond the scope of this research.

A. SUMMARY OF FINDINGS AND ANALYSIS

Based on the latency comparison chart (Figure 60), our research arrives at the conclusions shown in Table 2.

Table 2. Summary of Findings

ANUSD on Private IBM Hyperledger Fabric Network	**ANUSD on Private Ethereum Network**
1. The addition of miners to the private Hyperledger Fabric network does not seem to have a significant effect on the number of transactions per second. For all iterations, latency ranges from 3.4 to 3.7 transactions per second. This result occurs for all three tests (A, B, and C). 2. The average number of transactions per second using two peers in each organization ranges from 3.4 to 3.7.	1. The addition of miners to the private Ethereum network appears to have a significant effect on increasing the number of transactions per second in the tests involving ten iterations. Miners can handle more transactions per second in the network. 2. The average number of transactions per second using one miner and three peers ranges from 0 to 1. 3. The average number of transactions per second using two miners and three

ANUSD on Private IBM Hyperledger Fabric Network	ANUSD on Private Ethereum Network
3. The average number of transactions per second using four peers in each organization ranges from 3.5 to 3.7. 4. The average number of transactions per second using six peers in each organization ranges from 3.4 to 3.7.	peers ranges from 2 to 4 for the number of transactions above ten iterations. 4. The average number of transactions per second using two miners and six peers ranges from 6 to 8. The addition of peers affects the number of transactions per second that can be handled by the Ethereum network. The more nodes, the smaller the number of transaction queues that each node processes. Therefore, the processing load at each node is reduced and the system can handle transactions more quickly. The increase in TPS is greater in networks that have more peers.

As mentioned earlier, several obstacles were encountered during the testing of the implementation of ANUSD on both networks. These obstacles are described in Table 3.

Table 3. Obstacles Found during Testing

ANUSD on Private IBM Hyperledger Fabric Network	ANUSD on Private Ethereum Network
1. At the moment, Hyperledger Fabric's performance is largely dictated by implementation issues (API overhead, consensus algorithm implementations, capabilities for tuning, architectural overhead (Docker)) rather than theoretical limits. Hyperledger Fabric still lacks a fully robust consensus algorithm. IBM working towards Practical Byzantine Fault Tolerance (PBFT). The latest version (2.1.x) includes an algorithm that provides crash-fault tolerance (Raft) but not PBFT. Due to its flexibility (support for permission users and different consensus	1. Testing of the T2. AWS Medium is limited to CPU credit because T2 Medium has Burstable performance and has a CPU credit limitation. Hence, in testing, when the miner reached the CPU usage limit above the performance limit, the performance dropped to the point of 20% usage on the second miner. This greatly affected the TPS calculation for above 1,000 iterations in Tests B and C. 2. The test process for calculating latency by using the caliper on the Ethereum network did not work

ANUSD on Private IBM Hyperledger Fabric Network	ANUSD on Private Ethereum Network
algorithms) it theoretically provides far greater opportunity to optimize and achieve higher transaction rates, compared to Proof-of-Work schemes, because they allow the consensus algorithm to be tuned per the unique trust-topology of each business problem. 2. From this researcher's experiences, Hyperledger Fabric is very unstable. The system breaks down suddenly without warning. The problem is exasperated by the fact that the documentation and tools related to it are also quite poor. The documentation contains an abundance of information, but it is mostly theoretical, rather than practical. 3. The product is made considerably worse by IBM's architectural choice to base it on containers (Docker). In the long term this choice makes sense. It is easier, for example, once the Hyperledger network is stable, to deploy it in a 100-machine data center. But for now, the design hides everything inside containers, forcing the user to do everything through containers. Users cannot even run a single command-line argument directly. Instead, users must start a container (Virtual Machine (VM)) and execute the command in the VM.	smoothly. Several bugs were encountered. The caliper was not stable. Therefore, it was decided to use artificial test tools by using additional Python code. The test tool file is available in the installation package file.

From the summaries of findings and obstacles outlined in Tables 2 and 3, it can be concluded that U.S. Navy decision makers can use the results of this research as a guide and source of important input for building ANUSD in the U.S. Navy organization in the future.

B. RECOMMENDATIONS FOR FUTURE WORK

The research presented in this thesis on the implementation of ANUSD on a private IBM Hyperledger and a private Ethereum blockchain network is limited to a comparison of the latency block time (transactions per second) on the two networks. Due to limited time, this research has not been able to make similar comparisons of other factors.

Some possible future work that can be done to expand or supplement this research is compare security on these networks. Security is an important factor for consideration and requires an accurate comparison of results from the two networks in supporting the ANUSD to be built by the U.S. Navy. Such research will be a valuable reference for decision makers in determining which network to choose for the ANUSD of the U.S. Navy system.

The first security factor that can be explored as future comparative research is confidentiality in relation to the implementation of ANUSD on the two networks. Confidentiality is one component of the Cyber Security Triad and it supports security in implementing an application on an internet network. With confidentiality, we can ascertain whether the data sent between fellow nodes cannot be glimpsed by other parties who are not part of the core network.

The second security factor that can be explored as future comparative research is integrity in relation to the implementation of ANUSD on the two networks. Integrity is another component of the Cyber Security Triad. Applications used for an internet network must be ensured not to be vulnerable to Man-in-the-Middle Attacks. With this type of attack, the attacker not only compromises confidentiality but can also make changes to data sent from one node to another node. An attack on the network's integrity is much greater in severity as compared to an attack on the network's confidentiality. Therefore, the comparison of the integrity of the ANUSD on the two blockchain networks is an important area for future work and can also serve as a reference for U.S. Navy leadership.

Finally, availability is another security factor that can be explored in comparative research. Specifically, it would be valuable to compare the availability of the implementation of ANUSD on the two blockchain networks.

LIST OF REFERENCES

[1] F. Manik, “Analisa Parameter Ethereum pada jaringan peer to peer blockchain di aplikasi transfer koin terhadap aspek memory [analysis of Ethereum parameters on the peer to peer blockchain network in the coin transfer application on memory aspects],” *Proceedings of Enggineering—Public Knowledge Project*, vol. 6 no. 2, p. 1, 2019.

[2] R. Yasaweerasinghelage, M. Staples and I. Weber, “Predicting latency of blockchain-based systems using architectural modelling and simulation,” in *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden, 2017.

[3] U. W. Chohan, “A history of bitcoin,” working paper, UNSW Business School, Sydney, Australia,” 2017, p. 1. [Online]. Available: https://ssrn.com/abstract=3047875.

[4] C. Dannen, I*ntroducing Ethereum and Solidity*, New York, NY, USA: Springer, 2017.

[5] R.Bowden, “Block arrivals in the Bitcoin blockchain,” working paper, School of Mathematics and Statistics,” University of Melbourne, Victoria, Australia, 2018. [Online]. Available: https://search.proquest.com/docview/2071280019.

[6] P. Siriwardena, “The mystery behind block time,” Medium, 14 October 2017. [Online]. Available: https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a.

[7] BLOXROUTE, “A relay network for high performance,” BlockRoute, 2020. [Online]. Available: https://bloxroute.com/docs/bloxroute-documentation/architecture/relay_overview/. Accessed 24 February 2020.

[8] Hyperledger , “Hyperledger Fabric introduction,” October 2019. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html.

[9] Linux Foundation, “Measuring Blockchain Performance with Hyperledger Caliper,” Hyperledger.org, February 2018. [Online]. Available: https://www.hyperledger.org/blog/2018/03/19/measuring-blockchain-performance-with-hyperledger-caliper.

[10] E. Maras, “IBM shares how blockchain technology improves food supply chain transparency and efficiency,” Blockchain Tech News,July 2019. [Online]. Available: https://www.blockchaintechnews.com/articles/ibm-shares-how-blockchain-technology-improves-food-supply-chain-transparency-and-efficiency/.

[11] L. Hang, “A Novel EMR Integrity Management Based,” *Electronic MDPI*, vol. 1, Jeju National University, 2019.

[12] J.Poon, “Building a private ethereum consortium,” *Microsoft CSE Developer* blog, 1 June 2018. [Online]. Available: https://devblogs.microsoft.com/cse/2018/06/01/creating-private-ethereum-consortium-kubernetes/.

[13] Linux Foundation, “Hyperledger,” 2020. [Online]. Available: https://www.hyperledger.org/projects/caliper. [Accessed 28 april 2020].

[14] J. Benet, “IPFS-content addressed, versioned, P2P file system,” 14 July 2014. [Online]. Available: https://arxiv.org/pdf/1407.3561.pdf.

[15] L. Carlozo, “What is Blockchain,” *Journal of Accountancy*, vol. 224, no. 1, p. 29, 2017.

[16] M. Rauchs. &. G.Hileman, “Global Cryptocurrency benchmarking study,” Cambridge Centre for Alternative Finance Journal, vol. 33, pp. 33–113,” 2017. [Online]. Available: http://dx.doi.org/10.2139/ssrn.2965436.

[17] R. Merkle, “Authentication, and public key systems,” Ph.D. dissertation, Stanford University, Stanford, CA, USA, 1979.

[18] N. Shah, *Blockchain for Business with Hyperledger Fabric*, New Delhi, India: BPB, 2019.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California