NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

UTILIZING BLOCKCHAIN TO DESIGN AN EAST/WEST INTERFACE FOR FEDERATED SOFTWARE DEFINED NETWORKS

by

Scott C. Tollefson

December 2018

Thesis Advisor: Geoffrey G. Xie
Second Reader: Robert Beverly

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

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

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE UTILIZING BLOCKCHAIN TO DESIGN AN EAST/WEST INTERFACE FOR FEDERATED SOFTWARE DEFINED NETWORKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Scott C. Tollefson	
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)

After many years of development, Software Defined Networks (SDN) have started to see mainstream acceptance. Considerable work has established standards for the northbound and southbound interfaces of SDN controllers, but much work remains on the development of an east/west protocol. In this research, we take security, autonomy, and privacy as essential requirements for an east/west interface. We believe that the properties of permissioned blockchains, serializability, immutability, verifiability, and smart contracts, meet these requirements. Using Hyperledger Fabric as our blockchain protocol, we were able to develop a proof-of-concept for an east/west protocol on the ONOS SDN platform. We demonstrate successfully requesting a connection between separately managed SDN networks which leads to the installation of flows and data transfer between the networks. We further evaluate the impact of network latency on Hyperledger Fabric nodes to complete transactions; the results show a linear relationship between per hop network latency and transaction completion time.

14. SUBJECT TERMS blockchain, Hyperledger, networks, software defined networks, SDN, east/west interface	**15. NUMBER OF PAGES** 145
	16. PRICE CODE

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

UTILIZING BLOCKCHAIN TO DESIGN AN EAST/WEST INTERFACE FOR FEDERATED SOFTWARE DEFINED NETWORKS

Scott C. Tollefson
Lieutenant Commander, United States Navy
BS, North Dakota State University, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2018

Approved by: Geoffrey G. Xie
Advisor

Robert Beverly
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

After many years of development, Software Defined Networks (SDN) have started to see mainstream acceptance. Considerable work has established standards for the northbound and southbound interfaces of SDN controllers, but much work remains on the development of an east/west protocol. In this research, we take security, autonomy, and privacy as essential requirements for an east/west interface. We believe that the properties of permissioned blockchains, serializability, immutability, verifiability, and smart contracts, meet these requirements. Using Hyperledger Fabric as our blockchain protocol, we were able to develop a proof-of-concept for an east/west protocol on the ONOS SDN platform. We demonstrate successfully requesting a connection between separately managed SDN networks which leads to the installation of flows and data transfer between the networks. We further evaluate the impact of network latency on Hyperledger Fabric nodes to complete transactions; the results show a linear relationship between per hop network latency and transaction completion time.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Acronyms and Abbreviations

ACI application-centric infrastructure

API application program interface

AS autonomous system

BASH Bourne again shell

BFT Byzantine-fault tolerance

BGP Border Gateway Protocol

BST binary search tree

CFT crash fault tolerance

CLI command line interface

CO central office

CORD central office re-architected as a datacenter

DHCP Dynamic Host Configuration Protocol

DoD Department of Defense

EVM Ethereum virtual machine

GUI graphical user interface

HaaS hardware-as-a-service

IXP internet exchange point

IoT internet of things

JSON Javascript Object Notation

LAN	local area network
MAC	Media Access Control
Mbps	Megabits per second
MSP	membership service provider
NoSQL	non-structured query language
NPS	Naval Postgraduate School
ONOS	Open Network Operating System
OSPF	Open Shortest Path First
PKI	Public Key Infrastructure
QoS	quality of service
REST	representational state transfer
RFC	Request For Comments
RIP	Routing Information Protocol
RTT	round trip time
SDK	software developer's kit
SDN	software defined network
SDX	software defined IXP
SFP	SDN Federation Protocol
SQL	Structured Query Language
TCP	transmission control protocol
TSS	time-stamping service
USN	U.S. Navy

UDP	User Datagram Protocol
USG	United States government
VLAN	Virtual LAN
VM	virtual machine
XFT	cross-fault tolerance
YAML	YAML Ain't Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank Dr. Geoffrey Xie and Dr. Robert Beverly for their advice, support, and council. Their support brought clarity and focus and kept me on track to finish. Many thanks to Dr. Xie for being flexible to all my unannounced visits to his office for guidance!

I would like to thank the members of my cohort, Garret Walton, Nick Davis, Timberon Vanzant, and Vince Amos, for being a punching bag to my ideas and for their engaging discussions. Finally, I want to thank my family for their flexibility during my time at NPS.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

After many years of development, software defined networks (SDNs) is starting to see mainstream acceptance with projects like central office re-architected as a datacenter (CORD) [1] and Google's B4 [2]. Early work in SDN standardized SDN and SDN-enabling protocols on the southbound interface. Northbound interfaces have shifted to the development of representational state transfer (REST) application program interfaces (APIs) which are used by many of the active SDN controllers like Nox [3], OpenDaylight [4], Open Network Operating System (ONOS) [5], Ryu [6], Beacon [7], Floodlight [8], and OpenContrail [9]. The east/west interface, which enables communication between controllers, has had some development with protocols like SDNi and software defined IXP (SDX). Le et al. propose and define requirements for an east/west protocol called SDN Federation Protocol (SFP) [10]. SFP needs to be efficient and scalable, make use of metric information across networks, be autonomous and stable, and maintain privacy [10]. In this research, we take SFP's requirements to be essential for an east/west protocol. SFP was proposed to utilize a publish-subscribe model to prevent the protocol from having to handle every combination of options in the packet header [10]. The development and implementation of SFP were left by Le et al. as future work.

Meanwhile, blockchain protocols moved from simple cryptocurrency systems to complex architectures with the introduction of on-chain storage and chaincode/smart contracts. Blockchains have found uses in financial markets, internet of things (IoT), supply chain management, smart energy grids, etc. Only recently have blockchain protocols found use cases in networking. Raju et al. were able to introduce a network access provisioning for cognitive cellular networks using the blockchain protocol Ethereum. Raju et al. were able to reduce provisioning time by up to four times, decrease network signaling traffic by almost 40%, and reduce payment settlement by almost three times [11]. The properties of blockchains are serializability, immutability, and verifiability without implicit trust [12]. Serializability ensures that processes take place in order ensuring the correctness of concurrent operations such as transactions. Immutability ensures that once created, objects cannot be changed. Finally, verifiability without implicit trust means that operations can be

verified as correct without having to trust a single service performing the validation.

1.1 Problem Statement

In light of Le et al.'s requirements for SFP, we note that existing SDN east/west protocols do not meet all of the requirements. Google's B4 protocol was developed to connect data centers connected by Google [2]. B4 was not developed to be used between separately managed SDN networks and does not ensure privacy. The SDNi protocol was designed to facilitate inter-SDN routing by coordinating flows and sharing path, quality of service (QoS), bandwidth, latency, and other networking data. However, SDNi was designed to operate between SDN networks controlled by the same operators and does not take into account privacy [13]. Additionally, the Internet Draft for SDNi has expired [13]. SDX is another east/west protocol and is designed to be utilized at internet exchange point (IXP)s [14]. SDX interacts directly with Border Gateway Protocol (BGP) routers [14]. However, not all SDN networks connect directly with BGP routers. SDX compiles policy rules to combine them into as few rules as possible to improve efficiency. However, SDX's inbound and outbound policy rules are replicated to all nodes making privacy an issue.

Therefore, much work remains on the development of an east/west protocol. In this research, we take security, autonomy, and privacy as essential requirements for an east/west interface, and explore the use of permissioned blockchains to meet these requirements.

1.2 Research Questions

The requirements for SFP would make it an ideal protocol for an east/west interface, but the development of SFP was left to future work. The properties of the blockchains would be excellent properties to apply to an east/west interface for SDN networks. To the best of our knowledge, no one has tried to implement a networking protocol for SDN networks using blockchains. This research aims to answer the following question:

- Can a proof-of-concept model for an east/west protocol be developed using blockchains to connect two separately managed SDN networks?

If a proof-of-concept can be developed we look to answer the following questions:

- Does the blockchain based east/west protocol meet the requirements laid out by Le et al.? Specifically:
  - Is the protocol efficient and scalable to large networks?
  - Does the protocol make use of metric information across networks?
  - Are sub-requests responded to quickly?
  - Is the protocol autonomous and stable?
  - Is the network able to keep sensitive information from another network?
- Are the properties of blockchains useful for an east/west interface?

1.3 Research Contributions

In this work, we answered our primary research question and made the following contributions:

- We were able to develop a proof-of-concept for an east/west protocol on the ONOS SDN platform using Hyperledger Fabric. We demonstrated successfully requesting a connection between separately managed SDN networks which lead to the installation of data flows between networks.
- We were able to evaluate the impact of network latency and show a linear relationship between network latency and Hyperledger Fabric transaction processing time.
- We provided suggestions for future work to further the development of the proof-of-concept and to benchmark the Hyperledger Fabric protocol.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 is an introduction to SDN networking, Mininet, and Open vSwitch. Blockchain technology is discussed starting with the time-stamping digital files and covering Bitcoin, Ethereum, and Hyperledger Fabric. Chapter 2 concludes by covering the related work of SFP and the benchmarking of Hyperledger Fabric. Chapter 3 overviews the high-level design and presents the basic components for building a proof-of-concept model. Chapter 3 continues with the high-level design requirements for Hyperledger Fabric, resource sharing, and the test-bed SDN network. Chapter 3 concludes with a discussion of the evaluation criteria for the proof-of-concept model. Chapter 4 covers the implementation of the proof-of-concept. Chapter 4

details how the SDN network was built and how the controller is implemented. Chapter 4 continues with details of constructing the Hyperledger Fabric and the functions of contract code. Chapter 4 concludes with details of the Control Application and its functions. Chapter 5 demonstrates a functioning prototype and presents an analysis of Hyperledger Fabric performance. Chapter 6 presents conclusions and future work.

CHAPTER 2:
Background

Chapter 2 will cover the background of the topics and components used in this thesis. A background on Software Defined Networks and the issue of interconnecting Software Defined Networks is presented. The next section covers a background in blockchains including time-stamping digital document, Bitcoin, Ethereum, and Hyperledger Fabric. The chapter concludes with a discussion on related work including the work on SFP by Le et al. and the benchmarking of Hyperledger Fabric by Thakkar et al.

2.1 SDN

Traditional networking takes a decentralized approach to manage packets traversing the network. Routers and switches will utilize various decentralized protocols such as Open Shortest Path First (OSPF) and Routing Information Protocol (RIP) to determine how an individual packet will get forwarded. Each of the different routing protocols has advantages and disadvantages including their ability to enact policy. The decentralized nature of non-SDN networks means there is little control over certain aspects of routing and switching unless additional protocols or routing/switching instructions are introduced. Protocols like Bellman-Ford and Dijkstra's Algorithm determine for each routing device which port to put a packet on to get it to its destination. If a network administrator wanted a packet to take a particular path each device on the network would need to be manipulated to achieve the result.

SDN networks take a different approach and utilize a centralized network control architecture [15]. From the perspective of the SDN network controller, there are three interfaces, the north, south, and east/west, see Figure 2.3. The southbound interface is used for communication between the OpenFlow switches and the controller. For the OpenFlow protocol, communication traffic on the southbound interface is defined in the OpenFlow specification [16]. We will not cover all of the OpenFlow messages here but, we will briefly discuss the Packet In, Packet Out, Modify-State, Flow-Removed, and Hello messages [16].

- Hello is a message used by an OpenFlow switch and the controller to set up a

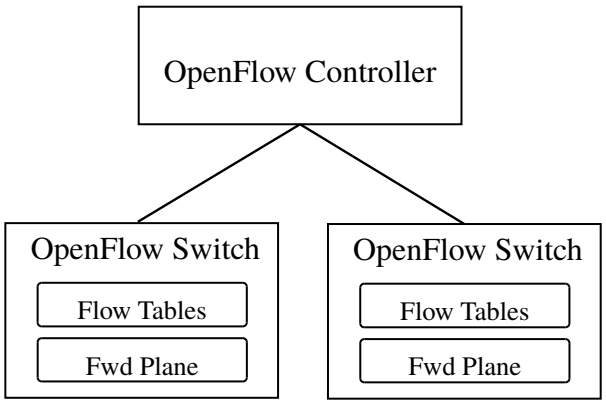


Figure 2.1. A simple OpenFlow network with two OpenFlow switches. Image adapted from [15]

connection between the controller and a switch.

- Packet In is a message used by an OpenFlow switch to send a packet to the controller for processing.
- Modify-State is a message sent by the controller to an OpenFlow switch after deciding how the packet should be handled. The OpenFlow switch will use the Modify-State message to update the flow table of a switch.
- Packet Out is a message that contains the packet that has been sent to the controller for processing. Packet Out is used to return a packet to the switch that the packet originated. Once it has returned, a Packet Out message can jump directly to the action set stage of the packet processing. A packet returned to a switch using Packet Out will generally be output to the table for processing through the new flow entry sent in the Modify-State message.
- Flow-Removed is a message sent to the controller by the OpenFlow switch to report that a flow has been removed. An installed flow can be removed at the request of the controller due to a hard timeout, such as the install duration expiring, or due to a soft timeout, such as no matches in a specified time. The Flow-Removed message includes the reason it was removed, the installed duration, the packet count, and byte count. The packet count and byte count can be used by an appropriately programmed controller to collect and display metrics about the network. To ensure time-accurate metrics get collected, flows can be installed with short duration so that data from Flow-Removed messages are collected more often.

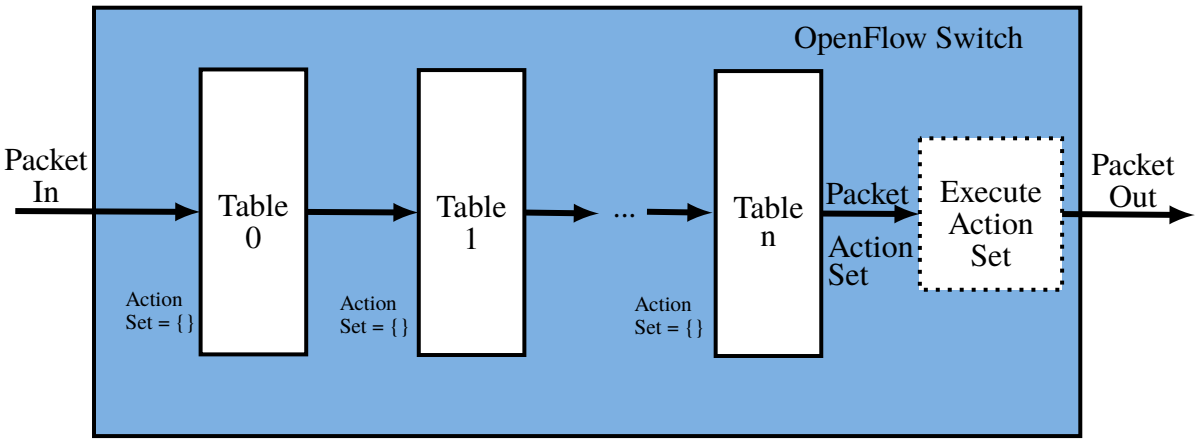


Figure 2.2. OpenFlow Packet processing through multiple tables. Each table has its own Action set. At each table the highest priority match will be found, action instructions are applied, metadata is updated, and the packet is forwarded to the next table unless the action set from the previous table stops the process. For example, the packet is output to a port. Image adapted from: [16]

A northbound API is used by applications and services to communicate with the management functions of the SDN controller. There is no universal standard for the northbound API interface but, many controller families have developed REST APIs for their northbound interface [17].

Finally, the east/west Interface is used by SDN controllers to communicate between controllers. East/west interfaces can be used on an internal network so multiple controllers can replicate states for redundancy. Additionally, east/west interfaces can be used between controllers in separate networks to negotiate traffic between networks.

2.1.1 Open vSwitch

Open vSwitch is a virtual switch created by the Linux Foundation that supports OpenFlow protocols [18]. Open vSwitch is a production quality virtual switch used in virtualized environments [18]. Open vSwitch can be used to create a network of virtual switches that can be used for creating virtual SDN networks. Once Open vSwitch is installed, a switch can be created using ovs-vsctl utility and flows can be installed using ovs-ofctl utility. Using the Open vSwitch commands, we can emulate a full SDN network. However, creating

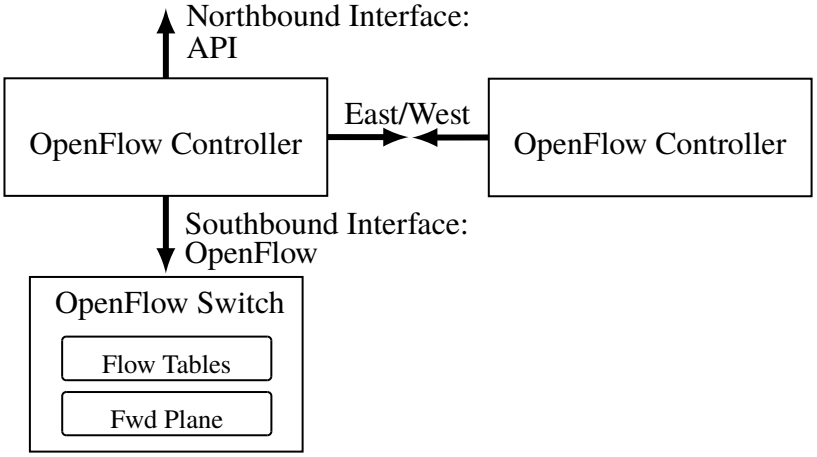


Figure 2.3. North, east, west, and south interfaces on an SDN controller.

a full network using the ovs-vsctl and ovs-ofctl commands would be quite cumbersome. Additionally, we do not have a way to simulate hosts on the network and would need to launch virtual machines (VMs) to function as hosts.

2.1.2 Mininet

Mininet is a virtual network emulator written in python and enables the rapid prototyping of SDN networks. Mininet makes creating Open vSwitch switches, hosts and links between switches and hosts much easier than using Open vSwitch alone. The hosts emulated by Mininet run a real kernel so each host has a command-line which can run any installed application on the host machine. Mininet has built-in network topologies such as linear, single, torus, and tree for rapid prototyping and can run custom typologies defined by the user in a python script. After Mininet is installed a network can be built using:

```
sudo mn --controller=default --link tc,bw=10,delay=10ms --switch=ovs --topo=linear,5
```

The above command will create a network with a linear topology with five Open vSwitch switches with one connected host per switch, see Figure 2.4. Each link will have a bandwidth of 10Mbps and have a latency of 10ms.

A custom topology can be defined in a python file. An example adapted from [19] is shown in Appendix A.1

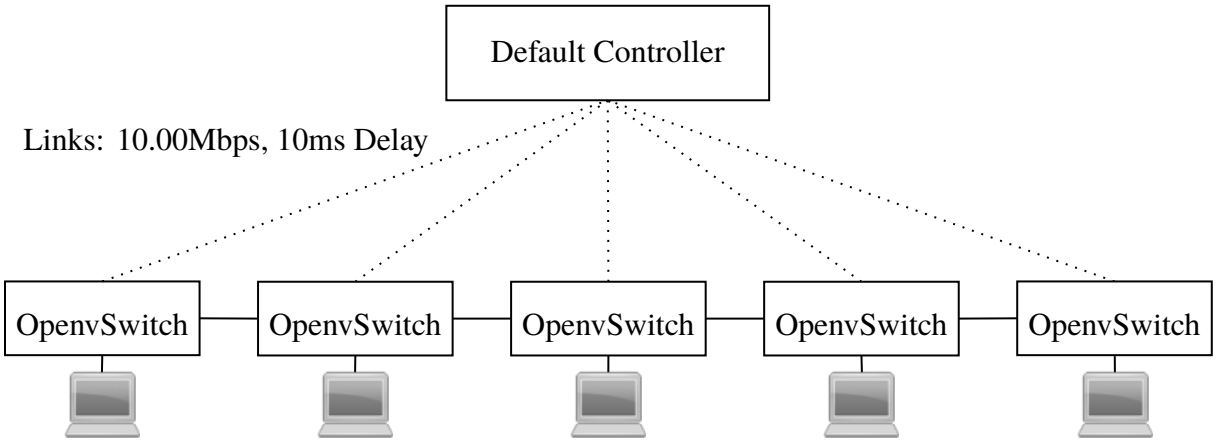


Figure 2.4. A simple OpenFlow network built with one command using Mininet.

The code for the custom topology can be run using the command:

```
sudo mn --custom <path><topologyname>.py
```

When executed, a network will be created with three switches (lines 14-16) with two hosts per switch (lines 19-21). Instead of the default controller, a remote controller at 172.17.0.2 listening on port 6633 is used (line 11). By using combinations of addHost, addLink, addSwitch commands we can build any custom topology up to the limits of the virtual machine hosting Mininet and connect the network to any available SDN controller. Once our network is running with a controller, we can bring up a terminal for any of the hosts and run a server or use any available command like iperf and ping to test the network connections.

2.2 Blockchains

A blockchain is an immutable electronic ledger that records transactions. Blockchains are used on distributed networks with no central authority or single point of failure. Nodes on a blockchain network use various protocols to come to a consensus on the current state of the blockchain. This section will discuss the concept of blockchains and the Bitcoin and

Ethereum protocols which will be used as a basis of comparison. The section will end with a discussion of Hyperledger Fabric, a permissioned blockchain protocol that uses a distributed trust system that does not require broadcasts of transactions to all nodes on a network.

2.2.1 Time-Stamping Digital Documents

Haber and Stornetta created a system to time-stamp a digital document by using a time-stamping service (TSS) [20]. A TSS takes a cryptographically secure collision-free hash function and produces a digest of fixed length from a digital document. A date and time are added to the digest, and the TSS digitally signs the message with a private key. A digital signature is used to identify the signer uniquely. The message and signature can later be verified to have been signed by the TSS by utilizing the TSS's public key. To perform verification that the original document was time-stamped, the document can be hashed and the hash value matched against the hash value produced, time-stamped, and signed by the TSS. Assuming that the hash function is computationally secure, users of the system must depend on the TSS being a trusted service.

To ensure that the TSS can never produce a fraudulent time-stamp of a digital file, Haber and Stornetta introduced a linking system [20]. When a client requests the TSS to time-stamp digital file x, the client will pass (y_n, ID_n) where y_n is the hash value of the n^{th} digital file and ID_n is the identification of the client making the n^{th} request. The TSS responds with signature $s = \sigma(C_n)$ where σ is the signing function and C_n is the n^{th} certificate containing the certificate number n, the time t_n, identification of the client requesting the signature ID_n, the hash value of the n^{th} digital file, and linking information L_n which comes from the previous certificate. After the $(n + 1)^{th}$ request has been processed, the TSS will send the client making the n^{th} request ID_{n+1}. Now, the client has (s, ID_{n+1}) which is attached/associated with file x. If the time-stamp of file x is challenged, the challenger can verify all of the information in s, then check forward in time by checking with client ID_{n+1} and requesting and verifying (s', ID_{n+2}). Additionally, the challenger can check backward in time by checking with client ID_{n-1} which is contained in L_n in s and so on. Implicit trust of the TSS is no longer required. However, if a corrupt TSS generates fraudulent signatures, the challenger must verify forward/backward in time far enough to detect them.

To avoid the use of a TSS altogether, a client can send their request for verification (y_n, ID_n) to k randomly generated client IDs. The final verification of the time-stamp of document x will be $[(y, ID), (s_1, s_2, ..., s_k)]$ where s_j is the signature of the j^{th} randomly selected client. Here, each selected client is performing the work the TSS was performing. To fake a verification, the client would need to find k clients willing to collaborate. However, if the clients were randomly selected using a random generating function and seeding it with y then the client cannot guarantee the selection of the k clients. If k is selected to be sufficiently large, then even a large majority of the clients being corrupt would not guarantee a selection of all corrupt clients. This scheme forms a distributed trust and does not require a centralized TSS to time-stamp files.

2.2.2 Bitcoin

The person or persons known as Satoshi Nakamoto used the idea of linking time-stamps with a distributed trust and used it to implement an electronic cash system called Bitcoin [21]. Users of Bitcoin can send digital payment transactions from one account to another. Without a centralized trust there needed to be a mechanism to ensure the proper ordering of transactions. To prevent race conditions, all transactions are made public (broadcast) and a proof-of-work system of verification and consensus was implemented [21]. Bitcoin's proof-of-work system starts with a genesis block which sets the initial condition of the system. With the genesis block established, a user on the system can execute a transaction which is broadcast to all nodes on the network. The transaction is placed into a block, which consists of a hash of the previous block (in the case of the first transaction, the previous block will be the genesis block), a nonce, and the transaction(s), see Figure 2.5. For a block to be valid, the hash is required to be less than the value of the networks difficulty value [21]. If the hash of the block is not less than the difficulty, the nonce is incremented, and the block is re-hashed [21]. Generally, a large number of hashes must be calculated before finding a hash that meets the difficulty value, requiring a significant amount of computing resources. The difficulty value changes over time and is based on the time it takes for 2,016 blocks to be processed [22]. If 2,016 blocks take less than 10 minutes to process then the difficulty is increased, and the next set of blocks will take longer to process [22]. A node on the network that collects transactions and searches for block hashes under the value of the difficulty is called a miner [22]. Since miners are performing work on the Bitcoin network, they are awarded Bitcoins for being the first to find an acceptable hash value [22]. A completed

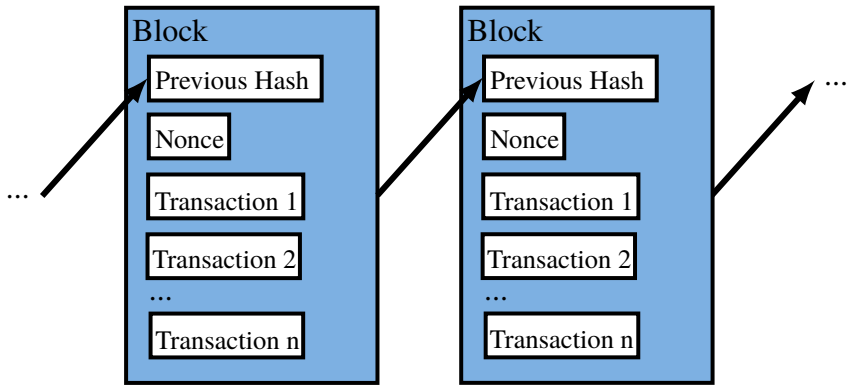


Figure 2.5. A simplification of a Bitcoin block of transactions which includes a Nonce and the hash of the previous block. By including the previous hash, a chain of blocks or blockchain is formed. Image adapted from [21].

block is broadcast to all nodes in the network where the transactions are verified by other nodes on the network [21]. The block is accepted when other nodes begin to use the hash of the block as the previous hash in the next block [21].

Nodes on the Bitcoin network can be full nodes which carry a copy of the entire blockchain or partial nodes where they only carry part of the blockchain [22]. Bitcoin uses a consensus method so that the longest blockchain is the most up-to-date [22]. When a new node is connected to the network, it will request each block from its peers and verify the hash [22]. The node cannot begin to verify transactions until it has all of the blocks [22]. As of this writing, the current size of the full blockchain for Bitcoin is 171.3 Gigabytes [23].

The probability p of finding the correct hash value from a single hash shown in equation 2.1, where D is the current difficulty of the Bitcoin network [24].

$$p \approx \frac{1}{D \times 2^{32}} \tag{2.1}$$

As of this writing, D is currently at $7,019,199,231,177$ [25]. The expected number of hashes to find a block is $D2^{32}$ [24]. Since the difficulty is adjusted based on a 10-minute time frame, to calculate a correct hash in 10 minutes based on the expected value will require 5.024×10^{19} hashes per second or 50 exa-hashes per second.

A challenger to a transaction can quickly do verification by re-hashing the block to verify it against the difficulty. An attacker cannot change a transaction without performing the

computational work to produce the correct hash [21]. If an attacker wanted to modify the results of a transaction, they would need to re-hash the block and all blocks that have come after the block they want to modify.

2.2.3 Ethereum

The Ethereum blockchain protocol builds upon the ideas of the Bitcoin protocol by adding contract code referred to as smart contracts, written in a Turing Complete language called Solidity, to be executed on the Ethereum virtual machine (EVM) [26]. Smart contracts execute state transitions on accounts. Individual accounts in Ethereum are made up of four fields, a nonce, ether (cryptocurrency) balance, contract code, and storage space [26]. The account storage in an Ethereum account is a key/value storage which persists between transactions.

Ethereum transactions can be initiated by account users or by a smart contracts [26]. Transactions contain a signature of the sender, a transaction value, data, and gas. Gas is a fee that is charged for each transaction, and the execution of smart contracts requires a gas charge per computational step [26]. Gas is used to pay for the work of processing the transaction, incentivize miners, and to prevent infinite loops in contract code. When the gas provided for a transaction runs out during the execution of the contract, the code will cease, the state will revert to the state before the transaction (minus the gas).

The addition of smart contract and account storage allows Ethereum to be a platform for some beneficial applications such as any type of financial contract, identity and reputation systems, computation markets, and distributed file storage. As an example, a simple messaging system can be developed where a smart contract can allow external accounts to write messages to the storage on another account. The contract can prevent external accounts from deleting or modifying messages but allow account owners to delete or modify the messages that are sent. As we will see later in this work, we can use a system like this simple messaging system to store and share information that can be used by SDN networks to come to agreements about network connections and share metric information.

Despite the potential that Ethereum has with the introduction of account storage and smart contracts, Ethereum suffers from many of the same negatives as Bitcoin. Because Ethereum uses a proof-of-work model similar to Bitcoin, it requires miners to repeatedly hash blocks

wasting computational resources and energy. Similarly to Bitcoin, transactions on Ethereum are not private and are broadcast across the network [26]. However, consortium blockchains and Private blockchains can be developed where transactions are only shared between members. Sizeable and complex contract code would be expensive to execute because of the Gas costs per computational step. Like Bitcoin, Ethereum nodes can be partial or full nodes with full nodes carrying a copy of the full blockchain. The size of the Ethereum blockchain has grown large over time and as of this writing is approximately 94 Gigabytes [27].

2.2.4 Hyperledger Fabric

Hyperledger is an open blockchain project developed under the Linux Foundation [28]. There are several frameworks under the Hyperledger project including Burrow, Fabric, Indy, IROHA, and Sawtooth. We will focus on the Hyperledger Fabric framework because it is the most general purpose framework. Hyperledger Fabric is a permissioned blockchain that has improvements to privacy and scalability over the proof-of-work blockchain models [28]. Permissioned means that the identities of the users are known to a degree and membership may be limited. Hyperledger Fabric uses smart contracts called chaincode similar to Ethereum [28]. When a transaction takes place, the transaction data is stored in a blockchain, and a local database called the state database is updated to reflect the current state of the system. There are three types of entities on the network used to carry out the model: clients, peers, and orderers [28]. Peers execute chaincode and validate transactions [28]. The orderer(s) are entities on the network that receive transactions and come to a consensus on the order of transactions [28]. Hyperledger Fabric uses channels for the communications between peers and orderers on a network [28]. Each channel will have a ledger and blocks are only provided to peers with verified channel membership [28]. Using channels allows for confidential transactions between peers and prevents off-channel peers from having to store data for transactions for which they have no interest. A peer can be a member of multiple different channels simultaneously.

Bitcoin and Ethereum both use an order-execute model which requires all transactions to be processed in series to prevent race conditions and double spending [29]. Hyperledger Fabric uses an execute-order-validate transaction model [28].

In the execute phase, a client initiates a transaction by sending the request for a transaction to the peer nodes. Peers execute the transaction (i.e., execute the chaincode), check for correctness, and endorse the transaction by signing it. Not all peers need to perform the execute and endorsement (signing). The determination of which peers are involved in a transaction depends on the current Endorsement Policy. The Endorsement Policy is a high-level policy and can be in the form: Peers X, Y, and Z are required to endorse transactions of chaincode K [28]. The endorsement policy can be tailored to the blockchain network to create an optimal balance between security and efficiency. After endorsing a transaction, the peers will return the transaction to the initiating client.

In the order phase, the endorsed transactions from peers are forwarded to the orderer(s) by the initiating client [28]. The orderer(s) are entities on the network that receive transactions and verify the order of the transactions based on the consensus. The orderer does not have a copy of the ledger, the state database, or the chaincode [28]. By keeping the orderer's function as simple as possible allowed Hyperledger Fabric to be the first blockchain protocol that allows for the selection of a consensus protocol [29]. Different consensus protocols such as Byzantine-fault tolerance (BFT), cross-fault tolerance (XFT), and crash fault tolerance (CFT) do not perform the same in all environments [30]. Hyperledger will allow developers to select a consensus model which best balances the security and efficiency for the blockchain network. Once the orderer has ordered the transactions, it will create and distribute blocks back to the peers. The distribution of blocks can be direct to peers or via a gossip dissemination protocol. Generally, there will be few orderers and many peers, so the gossip dissemination protocol improves scalability for large networks [29]. Hyperledger Fabric requires at least a single orderer in order to function. However, using only one orderer would not provide fault tolerance and nodes on the network would need to trust the orderer implicitly.

The validate stage consists of peers checking the endorsement policy and verifying keys. Each peer on the network will receive a block from the orderer and will individually validate the block. If the validation checks fail, the block is discarded by the peer. Hyperledger Fabric's blockchain ledger is an append-only blockchain, and each peer maintains a copy of the ledger as well as a copy of the state database. When the validation by the peers is complete, the ledger will be updated, followed by the state database. There are several implementations of the state database possible including LevelDB and CouchDB. LevelDB

is used for key-value pairs, and CouchDB can also store key-value pairs as well as well as Javascript Object Notation (JSON) formatted files.

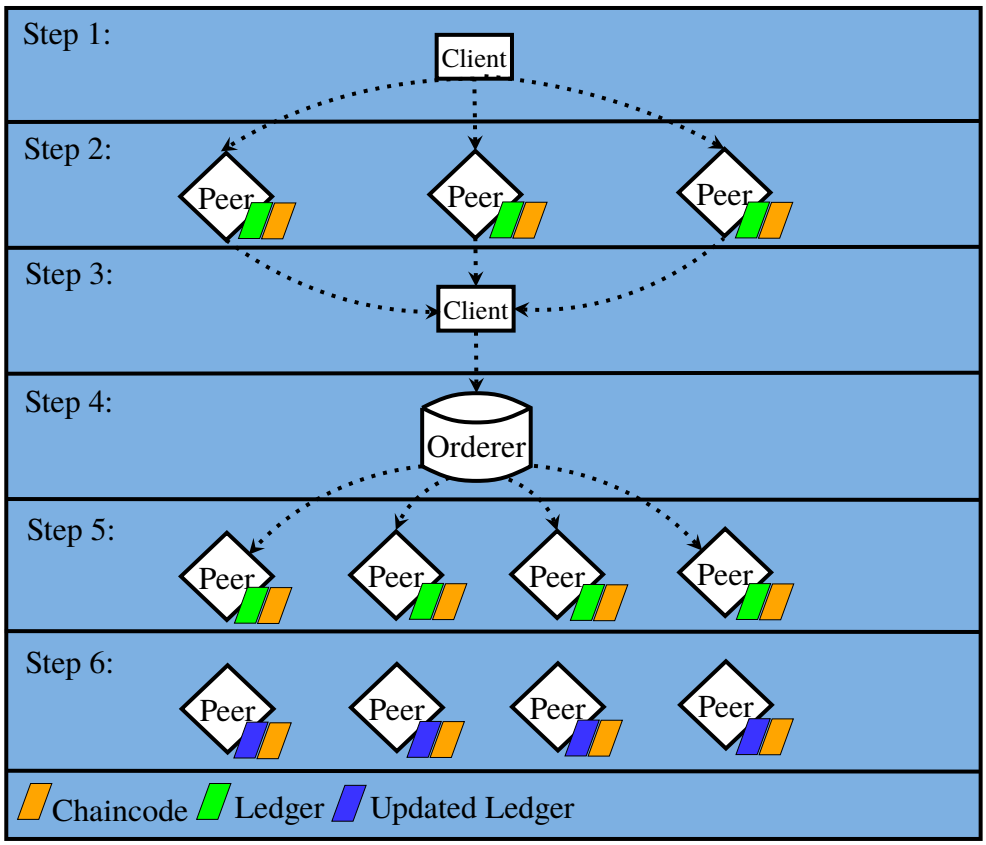


Figure 2.6. An example of the steps of the Hyperledger transaction process. Image adapted from [29].

An example of execute-order-validate process is shown in Figure 2.6. In this example, there is assumed to be four peers and only one orderer.

- Step 1: The client makes a transaction proposal to the endorsing peers. In this example, only 3 of the four peers are required to endorse by the Endorsement Policy.
- Step 2: The endorsing peers have a copy of the ledger and the chaincode. The peers execute the transaction and return the simulated state change, all keys associated with the transaction, and the metadata like the transaction ID and signature.
- Step 3: The client receives all endorsed transactions and sends them to the orderer.
- Step 4: The orderer establishes the transaction order using the consensus protocol installed. The transaction is collected into Blocks and delivered to peers. Delivery

can be direct to a peer but, in large networks, there will usually be many more peers than orderers. For large networks, the optional gossip dissemination protocol can be used to distribute blocks.

- Step 5: Blocks are returned to all peers including the peers that did not endorse the transaction.
- Step 6: Peers validate the block. The validation consists of checking the endorsement policy and verifying keys. If the validation checks fail, the block is discarded. If validation is successful, the local copy of the ledger and the state database are updated.

By utilizing the execute-order-validate method, transactions do not have to be executed in series to prevent race conditions [29]. Peers can execute chaincode in parallel which can significantly improve scalability.

2.3 Related Work

This thesis research is focused on investigating solutions for creating an east/west interface for SDN networks using blockchains. In this section, we will explore the SFP proposed by Le et al. to create an east/west interface. Once a basic proof-of-concept is developed for creating an east/west interface using blockchains, it will be compared to the requirements set by Le et al. for SFP. Additionally, we will discuss the work of Androulaki et al. to benchmark the Hyperledger Fabric protocol against a Bitcoin clone and use it as a basis of comparison.

2.3.1 SDN Federation Protocol

Le et al. [10] propose a new protocol, SFP, to create an east/west interface between SDN networks. SFP, or a protocol like SFP, is necessary because currently utilized protocols, like BGP, only require an IP address for destination-based forwarding. If BGP were to be extended to accommodate SDN networking, it would require entries in the routing base for every flow. For just the basic 5-tuple <d_IP, s_IP, s_port, d_port, protocol_ID> this can be as large as $2^{32} \cdot 2^{32} \cdot 2^{16} \cdot 2^{16} \cdot 2^{8} \approx 2.02 \times 10^{31}$ unique entries and every node would be required to advertise the best route for every entry. To prevent having routing entries for large combinations of packet headers, Le et al. propose that SFP use a pub-sub model. As an example, Network *A* will first send a Packet/Flow subscription request with a metric like

reachability and bandwidth to Network B. Network B responds to all sub-requests it is able or willing to route and includes the bandwidth that it can support, see Figure 2.7.

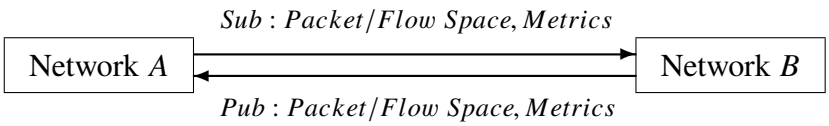


Figure 2.7. The proposed SFP message flow. Source [10].

Le et al. define three design qualities which SFP should have [10].

- SFP should efficiently use resources across networks. SFP should be efficient and scalable to large networks. The protocols should be able to make use of metric information across networks, and sub-requests should be responded to quickly.
- The SFP protocol should be autonomous and stable. When all networks implement their own policy, the conflicts in policies can lead to instabilities [31]. Universal algorithms can improve stability but prevent the use of individual network policies. SFP should use universal algorithms for stability but be flexible for networks to implement individual policy as much as practicable.
- Networks should be able to control privacy. A network should be able to keep sensitive information from another network.

Le et al. list several issues with SFP that would need to be addressed in future work [10]:

- Non-Adjacent Networks: SFP is discussed between two adjacent networks. The exchange and negotiation of flows between non-adjacent networks would introduce additional issues.
- Multiple Flow Queries: A domain would need to learn about the impact of multiple flows. If flows are queried individually, a response might show bandwidth available across a physical link. As the flows are implemented the bandwidth available will change, and the flows will not receive the bandwidth announced in the original query. To prevent negative impacts from individual flow requests, multiple flows should be allowed to be subscribed concurrently. However, querying multiple flows may allow a network to map out another network and compromise privacy.

- Correctness and Stability: The properties for correctness and stability need to be developed and tested.

We recognize there is other research into SDN federations and that SFP is one line-of-effort. This research will not attempt to implement the SFP protocol. However, we will attempt to utilize a blockchain protocol to perform a similar function. We will investigate if the requirements, as well as the issues Le et al. left for future work, can be addressed by utilizing the properties of permissioned blockchains.

2.3.2 Benchmarking Hyperledger Fabric

One of the crucial properties for SFP set by Le et al. is scalability. Since this research utilizes Hyperledger Fabric, we need to better understand if Hyperledger Fabric is scalable. Thakkar et al. performed benchmarking on the configurable parameters of Hyperledger Fabric [12]. The configurable parameters include block size, endorsement policy, channels, resource allocation, state database choice, and latency. In their testing, Thakkar et al. made several observations including [12]:

- Latency was slightly higher for a larger block size until a saturation point of 140 transactions per second was reached.
- Latency was slightly lower for larger block sizes when the transaction rate was higher than the saturation point of 140 transactions per second.
- For a given block size X, the transaction time increased if the transaction rate was below a specific value Y. The decrease is because the transactions must wait for a timeout before a block is issued. If the transaction rate is above Y, an increase in transaction rate will increase the rate that blocks are issued. Increasing the rate that blocks are issued will increase the transaction time because of the extra block validations.
- The transaction processing time increased linearly with the number of endorsing peers. The majority of the time increase was due the increased time in validating the signatures in a block. Each organization's membership service provider (MSP) only maintains its own organization's signatures. Signatures for outside organization require a request for validation from the outside organizations MSP.
- Increasing the number of channels increases the throughput and decreases latency but

increase the host machines CPU utilization.

- Transaction throughput was three times faster with the Go LevelDB than with the CouchDB.
- With the CouchDB, the endorsement time and state database update time increased with the number of lines to write in the transaction.

By combining their observations, Thakkar et al. were able to suggest and implement a number of optimizations to increase the transaction throughput over a single channel. The optimization leads to an increase the transactions per second from 140 to 2250 [12].

The optimizations made by Thakkar et al. will be significant when utilizing Hyperledger Fabric as a key component of an east/west interface for SDN networks. In particular, Thakkar et al. implemented a local cache of deserialized identities which reduced much of the cryptographic overhead and prevented a round trip time (RTT) to each signer's MSP for verification [12]. However, Thakkar et al. used a fixed latency on a 3 Gbps link when performing experiments [12]. The assumption was made that the network was not a bottleneck and network testing of Hyperledger Fabric was left to future work [12]. In this work, Hyperledger Fabric will be used as a networking protocol. It is important to extend the benchmarking performed by Thakkar et al. to understand better how Hyperledger Fabric performs at different network latencies.

CHAPTER 3:
Design

Chapter 3 will discuss the high-level design requirements for developing a proof-of-concept for an east/west protocol. In addition to the requirements, we discuss Hyperledger Fabric, resource sharing, our SDN network, and a Control Application. We conclude the chapter with a discussion on evaluation.

3.1 Requirements

The necessary components of our proof-of-concept will include a network emulator, network controller, blockchain protocol and a Control Application. On the blockchain network, peers executing transactions form a logical connection to other peers on the network. If we connect the peer nodes connected to a blockchain network to the northbound interface of the controller, we form a logical east/west interface between controllers via the blockchain network, see Figure 3.1. With this design, transactions on the network can instruct the controller to update flows. As an example, if the network on the left side of Figure 3.1 wanted to pass traffic across or to a host on the network on the right, an arrangement for this action can be made via contract code. Once the contract is executed, both networks can instruct the controllers to carry out the contract.

With this design, we should be able to connect:

- Adjacent SDN networks with separately managed controllers.
- Non-adjacent SDN networks with separately managed controllers and traditional routing between the networks.
- A series of separately managed SDN networks where traffic from one network must pass through another network to get to the destination network.

To test this concept, we will need:

- A blockchain network with multiple peers.
- Contract code which executes transactions
- A data structure on the blockchain to share resources information between networks.

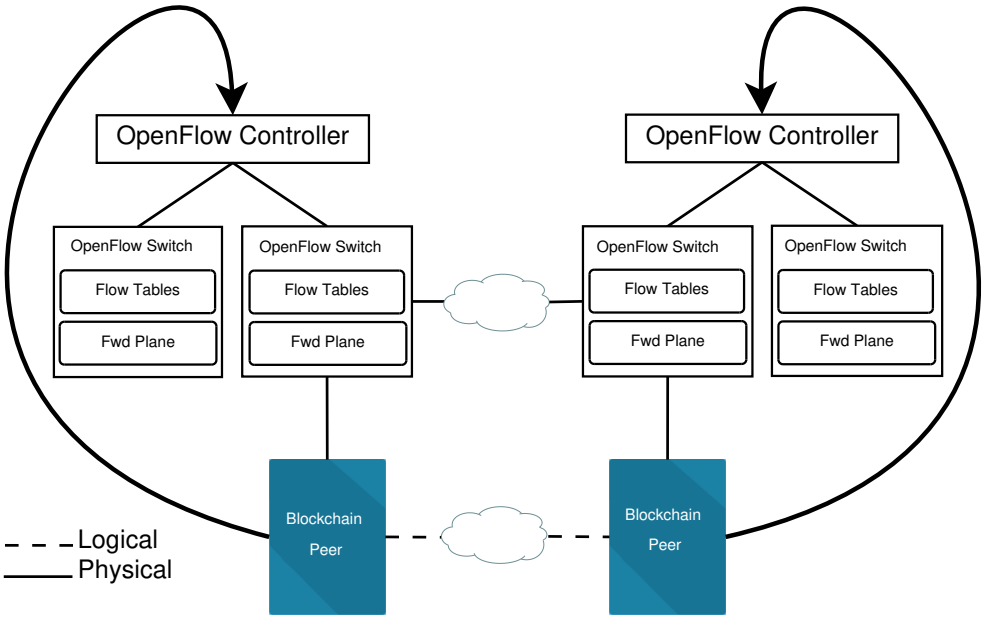


Figure 3.1. A logical east/west interface using a blockchain network.

- A northbound interface on a controller for adding and withdrawing flows.
- A minimum of two connected SDN networks.
- When transactions occur on the blockchain network, appropriate instructions will be generated and sent to the controllers northbound interface. Since forming a connection will not be part of any existing blockchain protocol, we will need an application that we call the Control Application to form this connection.

3.2 Hyperledger Fabric

Although a proof-of-concept could potentially be developed using Ethereum, the broadcast of all transaction across the network would violate the privacy requirement proposed by Le et al. Additionally, the way Ethereum operates violates the efficiency of resources requirement. For example, all nodes on the network process the smart contracts/chaincode and store every other node's transactions and miners repeatedly hash blocks for consensus.

Hyperledger Fabric was selected as the blockchain protocol because of its potential to process transactions much faster than other blockchain protocols. Hyperledger Fabric does not need to repeatedly hash blocks and can segregate transactions to private channels. Additionally, Hyperledger Fabric's consensus protocol can be selected for individual networks

allowing for a selection that is optimized for the transaction types that will be used on SDN networks. Androulaki et al. ran tests on Hyperledger Fabric and was able to achieve 3500 transactions per second with less than one-second latency on a single channel [29]. The actual time for transactions take to process will depend on many factors including the choice of consensus, number of nodes, network latency, and the types and size of the transactions.

Hyperledger Fabric's private blockchains/channels mirror the publish-subscribe structure suggested by Le et al. When a node updates their state information via a transaction (publish) the state is replicated to all nodes that have joined the same channel (subscribe). When a channel is joined between two networks a genesis block is created with the initial state of the network and populates the resource sharing data which are replicated to all nodes. Only nodes in the endorsement policy execute the transaction chaincode, and multiple queries of resource sharing data are handled using the local databases which are in keeping with the efficiency of resources requirement of Le et al.

Hyperledger Fabric provides resources for rapidly prototyping a Fabric network. Bringing the network online spawns the required elements namely the peers, clients, and orderer running in Docker containers. To test our design, transactions will only take place on one machine where latency between nodes is minimal and can be controlled. However, we are still able to test the establishment of flows initiated by transactions which will establish our proof-of-concept. A host on VM 1 will be set up to continuously send data to a host running on the SDN network on VM 4. The Hyperledger Fabric network will be set up on VM 4. A transaction will be initiated on the Hyperledger Fabric network on VM 4. After the transaction has completed, the Control Application will update the SDN controller on VM 4 from the information contained in a Hyperledger Fabric transaction and flows will be installed. Once the flows are installed, the connection between hosts on VM 1 and VM 4 will be established.

3.3 Resource Sharing

Following the requirements of Le et al., networks need to share resource and metric information across domains. The Hyperledger Fabric blockchain allows for key-value and JSON formatted data storage which can be used to store the resource sharing data we wish to implement. Hyperledger Fabric's JSON formatted non-structured query language (NoSQL)

database supports rich queries. As an example, rich queries cam find a network with available bandwidth above or below a specified value. Because the rich queries would prevent repeated searches for specific search values, we will focus the design on the JSON data structure using couchDB. For our proof-of-concept, we will focus the design on adjacent SDN networks with separately managed controllers. To demonstrate resource sharing, we will design the data structure to share available bandwidth and available hosts. Available bandwidth is a useful metric for any network managing flows, and the available hosts will be used to demonstrate connecting a network to an available compute resource.

3.4 SDN Network

To expedite development of a proof-of-concept the SDN networks will be developed in the Mininet emulation environment. The drawback of this approach is that all networked devices including controllers, routers, switches, hosts, peers, clients, and orderers will be running on the same host machine. A large and complex SDN network is not necessary for a proof-of-concept. However, a large scale network will be necessary for a complete measure of scalability and will be left for future work.

Any SDN controller with a northbound API will function for building a proof-of-concept. However, ONOS is a production-ready controller with a graphical user interface (GUI), a northbound API, applications, and the intent framework. The intent framework is a key feature because it translates high-level policies into the installation, modification, and upkeep of flows. Adoption of ONOS is likely to increase because ONOS is a critical component to CORD [1]. CORD is designed to replace closed-source, hardware and software at telecommunications central office (CO)s. A Survey by IHS Markit showed 85% of telecommunications COs intend to implement data centers and 70% of those intending to implement CORD [32]. ONOS uses the OpenFlow protocol, and although there are alternatives such as Cisco's application-centric infrastructure (ACI) there is either little adoption or the protocol is proprietary.

3.5 Control Application

Transactions and queries in Hyperledger Fabric can be initiated using commands on Hyperledger a CLI container connected to an anchor peer or on a client connected to an anchor

peer using the Hyperledger Fabric software developer's kit (SDK). Intents can be installed on the ONOS controllers using the ONOS REST API. For our prototype, we need an application to connect the execution of a transaction to the installation of flows which we call the Control Application. The Control Application should be notified when a transaction is completed that requires an intent to be installed and communicate with the ONOS controller to install the intent; see Figure 3.2.

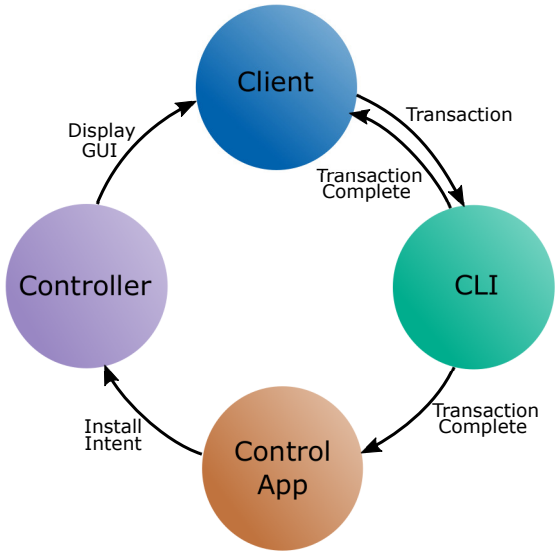


Figure 3.2. Control Application concept

3.6 Evaluation

To evaluate the proof-of-concept, we will establish flows between non-adjacent SDN networks with separately managed controllers and traditional routing between the networks by executing a transaction on the Hyperledger Fabric network. Once the proof-of-concept is established, data will be gathered on transactions under different transaction arrival rates and network latencies.

Because the Hyperledger Fabric is not fully optimized [29], BFT consensus is not complete, and our proof-of-concept will use VMs, large-scale testing will be left to future work. Additionally, developing a proof-of-concept for a series of separately managed SDN networks where traffic from one network must pass through another network to get to the destination network will require additional resource sharing and will be left for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Implementation

Chapter 4 will cover the components used for developing a proof-of-concept for developing an east/west interface using blockchains. The necessary components of the system will include a network emulator, network controller, and a blockchain protocol. Once the SDN test network is set up using Mininet with an ONOS controller, the blockchain protocol will be implemented using Hyperledger Fabric. A Control Application will connect to ONOS using the northbound interface and act as a client/application on Hyperledger Fabric, see Figure 4.1.

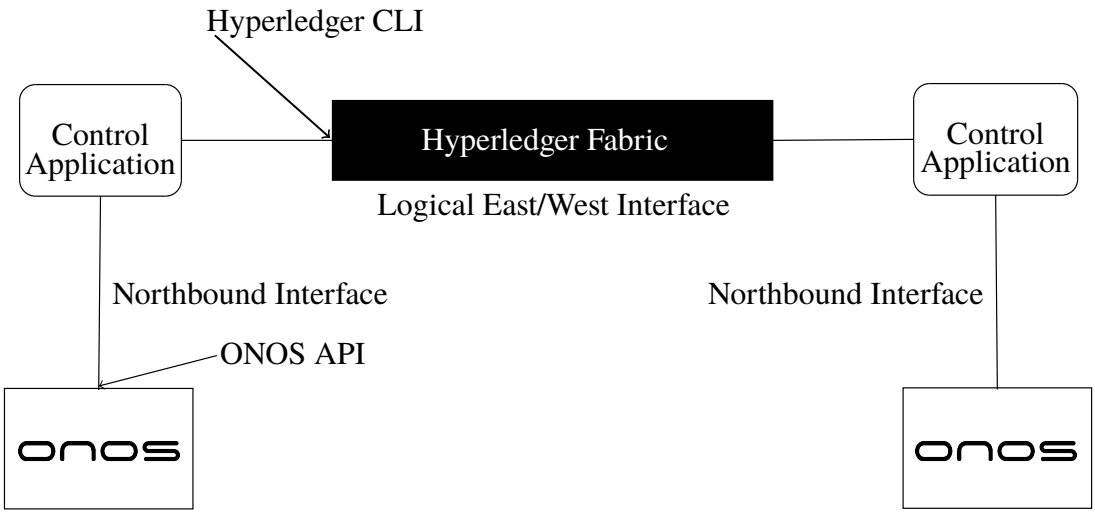


Figure 4.1. Creation of an east/west interface using the ONOS northbound interface and Hyperledger Fabric.

4.1 Network Implementation

The network consists of the network components, protocols, and controllers. OpenFlow was the first choice as the southbound protocol. Although there are alternatives to OpenFlow, many are proprietary. OpenFlow meets all of the needs of this project and is open-source with an extensive user-base and rich documentation [15]. OpenFlow capable hardware switches are available for expanding the research outside of virtualization. Mininet was

selected as the network emulator to create virtual networks that are compatible with OpenFlow [33]. Additionally, Mininet uses Open vSwitch, which supports OpenFlow. Hosts on Mininet can be used to emulate network traffic to other hosts running both server and client applications and creating realistic network traffic.

There are several drawbacks when using a virtual network for testing. All components of the network run on the resources of the machine hosting them. As the virtual networks get more complex, more load is put on the host machine's resources. Running multiple networks with applications on multiple hosts will drain computing resources making simulations slow, and impact the blockchain network and its transactions. Taking the compute resources into account, if the network is built with a small number of switches and hosts it can operate below 100% capacity and still produce a realistic emulation for a proof-of-concept.

The network built for this project is shown in Figure 4.2. The test network utilizes two separately managed SDN Networks connected by pfSense® virtual routers. All Virtual machines are running on the same host and hypervisor (VirtualBox). Traditional routers were included in the implementation to demonstrate connecting the SDN networks over traditional routing infrastructure. Connecting the SDN networks using traditional routers was done to demonstrate connecting the network over the internet. If the SDN networks were directly connected, they can be run on a single VM with two separate controllers. To connect the networks, all VMs are given an internal network connection in VirtualBox. The network connection is accomplished by setting the network adapter to Internal Network in VirtualBox and creating three separate networks, Inet1, Inet2, and Inet3. Inet1 is used to connect VM 1 and VM 2, Inet2 connects VM 2 and VM 3, and Inet3 connects VM 3 and VM 4. After logging into VM 1 or VM 4 the default interface, enp0s3, will be available for the inet connection between the VMs. The Mininet network needs to be attached to enp0s3 so that network traffic running on Mininet has a connection to the internal network adapter. The connection to enp0s3 is accomplished by using the ovs-vsctl utility in Open vSwitch by using the command:

```
sudo ovs-vsctl add-port [bridge name] [interface name]
```

The network interface can now acquire an IP address from the pfSense router using Dynamic

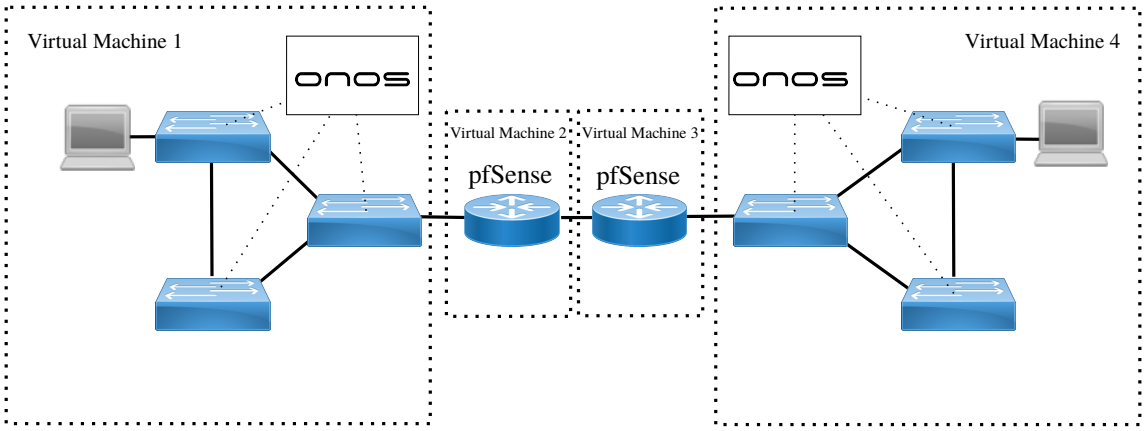


Figure 4.2. A virtual test network with separately managed SDN networks connected by traditional routers.

Host Configuration Protocol (DHCP). Getting an IP can be completed by forcing the interface to drop its default configuration and re-acquire an IP address:

```
sudo ifconfig enp0s3 0
sudo dhclient
```

The Mininet hosts are assigned a static IP address by default. The hosts can be forced to acquire an IP from DHCP by running the same commands as above. However, acquiring an IP on the hosts cannot happen until flows are installed because ONOS will drop all packets by default.

The code to build the network pictured in Figure 4.2 is included in Appendix A.2. After bringing up the network on VM 1 the same steps are repeated for VM 4.

The final piece of the network configuration is setting up the subnets between the VMs. The subnet between VM 1 and VM 2 was selected to be 10.10.10.0/24, the subnet between the pfSense routers (VM 2, VM 3) was selected as 20.20.20.0/30, and the subnet between VM 3 and VM 4 was selected to be 10.10.11.0/24. A static route was installed on both pfSense routers to ensure the subnets can connect.

4.2 Network Controller

ONOS was selected as the network controller for this project because of the rich feature set it offers coupled with active development and an active community of users. The ONOS controller uses applications written in Java to perform controller functions. An instance of an ONOS controller can run independently on a Docker container and can be clustered for scalability and fail-over redundancy. The Docker containers allow individual controllers to be started and shutdown independently of the other controllers in a virtual environment. ONOS has a well designed GUI for displaying network graphs, link status, link throughput, controller status, and node and switch status; see Figure 4.3.

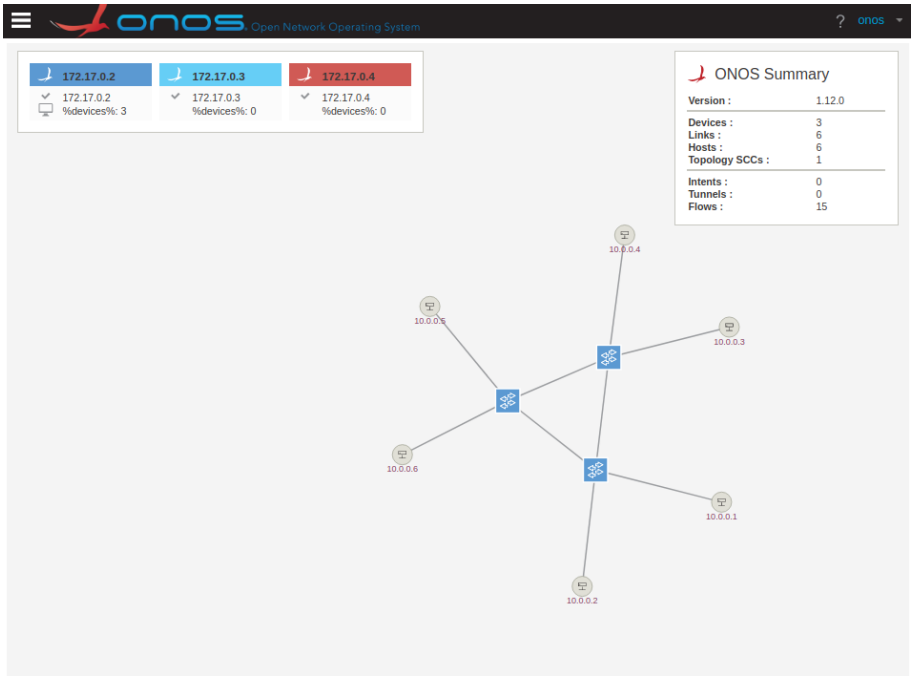


Figure 4.3. Image of the ONOS GUI depicting the custom topology in 2.1.2, depicting 3 switches with 2 hosts per switch utilizing a cluster of controllers.

The intent framework is a key feature of ONOS because it translates high-level policies into the installation, modification, and upkeep of flows [34]. By using intents, the Control Application needs to send a single message to ONOS from the information contained in a Hyperledger Fabric transaction instead of translating the transaction into multiple flow requests which need to be sent to all the active switches on the network. The intents, for example, allow for the user to specify flows between two points on the network. The intent framework will determine the path based on the criteria set by the user and install flows on

the switches to ensure the intent is met [34]. If conditions change such as a link going down or a link exceeding a bandwidth limitation, the intent framework will select and apply flow rules to ensure the conditions of the intent are met. These flow rule changes can include actions such as throttle bandwidth for lower priority flows or find alternate paths. An intent can be applied to the system using an ONOS application. When a packet arrives on a switch and does not have a matching flow entry installed, the packet is forwarded to the controller using the Packet In message. Once the controller receives the packet, if it matches criteria set by the operator in the ONOS application, an intent can be created. Additionally, intents can be installed by other programs and services using the northbound interface.

ONOS uses a REST API for the northbound interface. The API can accept a JSON formatted message on an open socket listening on http://<controllerIP>:8181/onos/v1/. A user or program can perform various queries or install intents, flows, vlans, and applications. As an example, the hosts on the SDN network on VM 4 in Figure 4.2 can be queried by sending a request to the ONOS northbound interface using the command:

```
curl -X GET --header 'Accept: application/json' 'http://172.17.0.2:8181/onos/v1/hosts' --user UNAME
 ↪   --pass PASSWORD
```

The response on a network with three hosts is a JSON formatted message:

```
{
  "hosts": [
    {
      "id": "00:00:00:00:00:01/None",
      "mac": "00:00:00:00:00:01",
      "vlan": "None",
      "configured": false,
      "ipAddresses": ["10.10.11.101"],
      "locations": [
        {
          "elementId": "of:0000000000000001",
          "port": "1"
        }
      ]
    },
    {
      "id": "00:00:00:00:00:02/None",
```

```
      "mac": "00:00:00:00:00:02",
      "vlan": "None",
      "configured": false,
      "ipAddresses": ["10.10.11.102"],
      "locations": [
        {
          "elementId": "of:0000000000000002",
          "port": "1"
        }
      ]
    },
    {
      "id": "00:00:00:00:00:03/None",
      "mac": "00:00:00:00:00:03",
      "vlan": "None",
      "configured": false,
      "ipAddresses": ["10.10.11.103"],
      "locations": [
        {
          "elementId": "of:0000000000000003",
          "port": "1"
        }
      ]
    }
  ]
}
```


To install an intent using the northbound interface a JSON formatted message can be sent to the controller using a PUT request, see Figure 4.4 for an example of the intent format.

Finally, ONOS collects metrics by collecting information from Packet In, Packet Out, Flow Mod, Flow Removed, Stats Request, and Stats Reply messages as well as system metrics from the host machine [34]. The collection of OpenFlow messages can provide network-wide metrics with little overhead [35].

4.3 Hyperledger Fabric

In this section, we will discuss the configuration files for Hyperledger Fabric, the installation of the environment, contract code, and channel setup, and the implementation of the chaincode.

```
{
  "intents": [
    {
      "type": "HostToHostIntent",
      "timeout": 1000,
      "id": "0x0",
      "key": "0x0",
      "duration": 100,
      "appId": "org.onosproject.gui",
      "resources": [
        "00:00:00:00:00:01/None",
        "00:00:00:00:00:02/None"
      ],
    }
  ]
}
```

Figure 4.4. A JSON formatted intent for the ONOS intent REST API. This intent is a Host-to-Host intent and will allow the host with the MAC address 00:00:00:00:00:01 on VLAN None to communicate with the host with the MAC address 00:00:00:00:00:02 on VLAN None. This intent will be installed with a duration of 100 seconds.

4.3.1 Fabric Configuration

To build the proof-of-concept and test the network, a Hyperledger Fabric test network will be created with all of the required network entities including peers, orderer, command line interface (CLI), and state databases. In this thesis, we will use modified versions of the Hyperledger example code provided in the install files [36]. The files include four YAML Ain't Markup Language (YAML) files for selecting the network and cryptographic configurations. The YAML files function as follows:

- crypto-config.yaml shown in Appendix A.3.1, contains a list of the nodes on the network and is used by Hyperledger Fabric's cryptogen utility to generate certificates for the organizations on the network. The normal process for generating keys is for an organization to utilize the MSP system to maintain and generate keys for an individual organization on the network [37]. For this project, we will use the cyrpto-config.yaml and the Cryptogen utility to generate all the keys we need for testing.
- docker-compose-cli.yaml shown in Appendix A.3.2, is used to generate Docker containers for the orderer, peers, and a CLI. The containers generated with the folders specified in the environment section (line 65) and will have access to the files of the

host operating system specified in the volumes section (line 80). Line 82 and 84 are modified to point to the /thesis/chaincode/ and /thesis/scripts where new scripts and chaincode are added to run the tests for the SDN network.

- docker-compose-couch.yaml shown in Appendix A.3.3 is used to set up Docker containers for the couchDB when the couchDB option is selected as the state database. When selected, couchDB replaces the default LevelDB (key-value) database and uses key-values as well as JSON formatted files [38]. Using JSON formatted data allows for creating easily readable data which can be quickly queried [38]. CouchDB uses a B-Tree data structure that has $O(\log n)$ insert, search, and delete operations. The couchDB Docker container's web-enabled interface can be accessed using the port defined on line 23 by accessing http://localhost:<port>.
- configtx.yaml shown in Appendix A.3.4 contains the profiles used by the configtxgen tool to create configuration artifacts for orderer genesis block, channel configuration, and anchor peers. Anchor peers are peers on the network connected to a Hyperledger channel and able to communicate with other peers from other organizations. The code is presented here unmodified from the source. Line 110 allows for selection of the orderer type with solo and Kafka as currently available options.

To bring the network online, we will use the shell script byfn.sh provided in the Hyperledger Fabric example files, see Appendix A.4.1. byfn.sh calls the cryptogen and configtxgen tools which utilize the YAML. Line 166 of byfn.sh is commented to prevent the networking from executing scripts/script.sh which is designed to run the Hyperledger Fabric tutorial demonstration. The network is brought online using the command:

```
./byfn.sh -m up -c sdnnetwork -s couchdb
```

The "-m up" option calls the networUp function on line 504 of byfn.sh. networkUp runs generateCerts, replacePrivateKey, and generageChannelArtifacts functions then runs docker-compose passing docker-compose-cli.yaml as an argument to create the CLI container. Additionally, because "-s couchdb" was included, docker-compose-couch.yaml is also passed as an argument. "-c sdnnetwork" names the private channel for the network sdnnetwork. The result of running the command is a series of Docker containers being spawned that run the peers, orderer, CLI, and state databases. The output from bringing the

network online will be:

```
Creating couchdb0 ... done
Creating couchdb2 ... done
Creating couchdb1 ... done
Creating couchdb3 ... done
Creating orderer.example.com ... done
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating peer1.org1.example.com ... done
Creating peer1.org2.example.com ... done
Creating cli ... done
```

Four peers have now been created with each peer having its own copy of the state database (couchDB). A CLI container which is used to run commands and scripts on the network. One orderer was created which does not have access to the state database.

To bring the network down use the command:

```
./byfn.sh -m down -c sdnnetwork
```

Once the network is up, the CLI can be accessed using the Docker command:

```
docker exec -it cli bash
```

4.3.2 Environment, Contract Code, and Channel Setup

After bringing the peers, orderer, CLI, and state databases online, we need to join the nodes to a channel and install and instantiate the chaincode. There are five scripts adapted from [39] that we will use to accomplish the setup.

- setclienv.sh shown in Appendix A.5.1, sets environmental variables for the channel name and the contract code name. sdnnetwork was used for the channel and thesis-chaincode references the file thesischaincode.go which contains a copy of the contract code.

- channel-setup.sh shown in Appendix A.5.1, establishes the sdnnetwork channel and joins all of the peers to the channel.
- install-chaincode.sh shown in Appendix A.5.1, must be called with one argument which is the version number of the contract code. When called install-chaincode.sh installs a copy of the contract code on the Docker containers of all peers.
- instantiate-chaincode.sh shown in Appendix A.5.1, is used to set the initial state of the blockchain. Like install-chaincode.sh, a version number is also passed as an argument and must refer to a matching version number of the installed chaincode.
- upgrade-chaincode.sh shown in Appendix A.5.5, is passed an argument for a version number and is used to install an updated version of the chaincode. The version number must not match any previously installed or upgraded version number.

The chaincode is ready to execute transactions after running:

```
source setclienv.sh
./channel-setup.sh
./install-chaincode.sh 1.0
./instantiate-chaincode.sh 1.0
```

After running the setup scripts, we are now able to execute the functions in the chaincode on the Hyperledger Fabric network. We can now use the "peer chaincode instantiate" command with the "-c" option which is the constructor argument. The constructor accepts a JSON formatted string. For example, we can now pass the string `'{"Args":["init","A","10","B","20"]}'` using the instantiate command. Passing a string will call an initialize function in the chaincode to create entries in the blockchain. The entities A and B are created, and a value for A is set to 10, and a value for B is set to 20. The values of 10 and 20 currently have no meaning, but we could decide that these values represent amounts of digital currency and we have just created a crypto-currency. To proceed, we need to write the data structure format for our resource sharing and additional functions for transactions and queries into the chaincode.

4.3.3 Contract Code

Hyperledger fabric supports contract code written in Go and Node.js. For this project, Go was selected over JavaScript because of familiarity with the language. The contract code

will consist of functions for carrying out various types of transactions and a data structure for resource sharing. Since we selected the state database to be couchDB we will utilize a JSON format for the data structure. In the Go language, we use a struct (structure) to define and store the data and use marshal to convert it to a JSON format for storage. When reading from the state database, we utilize the unmarshal function to convert a JSON string back to our struct. The struct developed for use in our proof-of-concept is:

```
type network struct {
        ObjectType  string    `json:"docType"`
        Name        string    `json:"name"`
        Address     string    `json:"address"`
        Hosts       int       `json:"hosts"`
        Availhosts  int       `json:"availhosts"`
        Bandwidth   float64   `json:"bandwidth"`
        Balance     float64   `json:"balance"`
        Pricepermb  float64   `json:"pricepermb"`
        Tenantslice []tenant `json:"tenantslice"`
    ASnum       int       `json:"asnum"`
    ASneighbor  []int     `json:"asneighbor"`
        TransCount  int       `json:"transcount"`

}

type tenant struct {
        ObjectType string    `json:"docType"`
        StartTime  int64     `json:"starttime"`
        Duration   int64     `json:"duration"`
        Bandwidth  float64   `json:"bandwidth"`
        Hosts      int       `json:"hosts"`
        SAddress   []string `json:"saddress"`
        TransNum   int       `json:"transnum"`
}
```

The network struct is used to store information about an individual SDN network. Critically, the data will contain the metric information used to make decisions about requests for flows from other networks. It consists of the following parts:

- "docType" is a string that uniquely identifies the type of object in the state database.
- "name" is a string that uniquely identifies the SDN network. In our example, the names could be peer0.org1.example.com or peer1.org2.example.com.
- "address" is a string that contains the network IP address for the SDN network.

- "hosts" is an integer and stores the currently available number of hosts. hosts will be used to set up an example where a network can advertise the number of hosts available for use in hardware-as-a-service (HaaS).
- "availhosts" is a string that stores a boolean value true, or false if the network has available hosts for use in HaaS.
- "bandwidth" is a double-precision floating-point used to store the total bandwidth in Megabits per second (Mbps) that the SDN network advertises as currently available.
- "balance" is a double-precision floating-point that represents a crypto-currency balance that is used to pay for network services.
- "Pricepermb" is a double-precision floating-point that represents a cost for reserving network bandwidth in Mbps.
- "tenantslice" is a list of tenant struct objects used to store information about networks with active contracts. The tenant struct is defined below.
- "asnum" is an integer used to identify the autonomous system (AS) where the SDNnetwork is a member.
- "asneighbor" is a list of integers that designated the neighbor AS's of a network.
- "transcount" is an integer that counts the number of transactions a network has made and uniquely identify the transactions.

At a minimum, the network struct includes information for requesting a flow for our proof-of-concept such as the network address and that bandwidth is available. Some fields like Pricepermb, asnum, and asneighbor were included to demonstrate additional types of information sharing.

When a network requests services from another network, the requesting network becomes a tenant of the hosting network. The tenant struct stores information about the contracts from the requesting networks and consists of the following parts:

- "docType" is a string that uniquely identifies the type of object in the state database.
- "starttime" is a 64-bit integer that stores the start time of the contract as a UNIX timestamp.
- "duration" is a 64-bit integer that designates the duration of an installed flow.
- "bandwidth" is a double-precision floating-point used to represent the supported bandwidth in Mbps that the contract requested.

- "hosts" is an integer and represents the number of hosts requested in the contract. hosts is used for a HaaS example.
- "saddress" is a list of strings. After a transaction, a flow will be establish for each saddress.
- "transnum" is an integer that uniquely identifies the transaction.

To carry out transactions and check the system state, several functions needed to be developed. The chaincode was designed with the following functions.

- Invoke Functions:
  - initNetwork takes a JSON formatted string and establishes the initial values of an account (network) in the state database. initNetwork returns a success or failure code.
  - readNetwork takes a "name" as an argument and returns a JSON formatted string with the current state of the requested network.
  - transactionNetwork takes a string as an argument. The string includes the requesting network name, requested network name, the number of hosts, bandwidth, and duration. Because transactionNetwork uses the PutState() function (see line 252 and 257 of Appendix A.6), it is the only function that generates a transaction which requires endorsement, ordering, and verification. All other functions are run locally.
  - pruneTenants takes a network name as an argument and searches the tenant list for entries that have exceeded the duration and removes them.
- Rich Query Functions: Rich Queries require couchDB as the state database and allow the passing of a selector. The selector includes an equality/inequality and value to query the JSON data structure. For example, the string "{"selector":{"docType":"network","hosts":{"gt":30}}}" will find the networks where the docType matches "network" and the number of hosts is greater than 30.
  - queryAvailhosts takes a string ("true", "false") as an argument and returns a list of networks with the searched value in "availhosts"
  - queryBandwidth takes an equality/inequality (gt/lt/gte/lte/eq) and a bandwidth value and returns a list of networks that match the search criteria.
  - queryPricePerMB takes an equality/inequality (gt/lt/gte/lte/eq) and a pricepermb value and returns a list of networks that match the search criteria.

- queryHosts takes an equality/inequality (gt/lt/gte/lte/eq) and the number of hosts, and returns a list of networks that match the search criteria.

- Blockchain search:
  - getHistoryForNetwork passes a network name to the function stub.GetHistoryForKey(). This function performs a search of the blockchain for the transaction history for the network named passed. Because the function performs a blockchain search, it is not fast. The actual speed of the function will depend on the size of the blockchain.

The full version of the contract code and its functions is presented in Appendix A.6.

4.3.4 Invoke and Transaction Tests

Before transactions can take place on the network we need to make some initial entries. To make entries we will use the invoke command with the initNetwork function. As an example, we will add network0 and network1 with the following command:

```
peer chaincode invoke -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
 ↪  /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ \
 ↪  example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C
 ↪  $CHANNEL_NAME -n $CC_NAME -c
 ↪  '{"Args":["initNetwork","network0","10.10.10.0","10","true","3579.28","41547","7.04","0","[24,
 ↪  34, 37]"]}'
peer chaincode invoke -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
 ↪  /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ \
 ↪  example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C
 ↪  $CHANNEL_NAME -n $CC_NAME -c
 ↪  '{"Args":["initNetwork","network1","10.10.11.0","10","true","4666.25","160307","2.19","1","[32,
 ↪  27, 30]"]}'
```

The entries can be verified using the readNetwork function. As an example, we can use readNetwork for the network1 entry created above:

```
Query Result:
{
  "address": "10.10.11.0",
  "asneighbor": [
    32,
    27,
```

```
    30
  ],
  "asnum": 1,
  "availhosts": "true",
  "balance": 160307.15208333332,
  "bandwidth": 4666.25,
  "docType": "network",
  "hosts": 10,
  "name": "network1",
  "pricepermb": 2.19,
  "tenantslice": [],
  "transcount": 1
}
2018-10-12 22:55:46.478 UTC [main] main -> INFO 003 Exiting.....
```

Additionally, we can view the entry in the couchDB by utilizing the web-GUI at http://localhost:5984/_utils/#database/mychannel_sdnnetwork/network1. See Figure 4.5.

Now that we have invoked entries for networks on VM 1 and VM 4 in Figure 4.2, we can process a transaction. As a test, network0 will request from network1 a flow to one available host to support two IP addresses, 10.10.11.101 and 10.10.11.102 for 25 seconds, with 10 Mbps bandwidth. The command to carry out the transaction is:

```
peer chaincode invoke -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
 ↪   /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ \
 ↪   example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C
 ↪   $CHANNEL_NAME -n $CC_NAME -c '{"Args":["transactionNetwork","network0","network1","25","10","1",
 ↪   "[10.10.11.101,10.10.11.102]"]}'
```

The result from network1 is an updated entry in the state database, specifically the transaction information requested shows in the tennantslice for network1:

```
{
  "_id": "network1",
  "_rev": "4-82243787d47c2e738433853ff220bee2",
  "address": "10.10.11.0",
  "asneighbor": [
    32,
    27,
    30
```

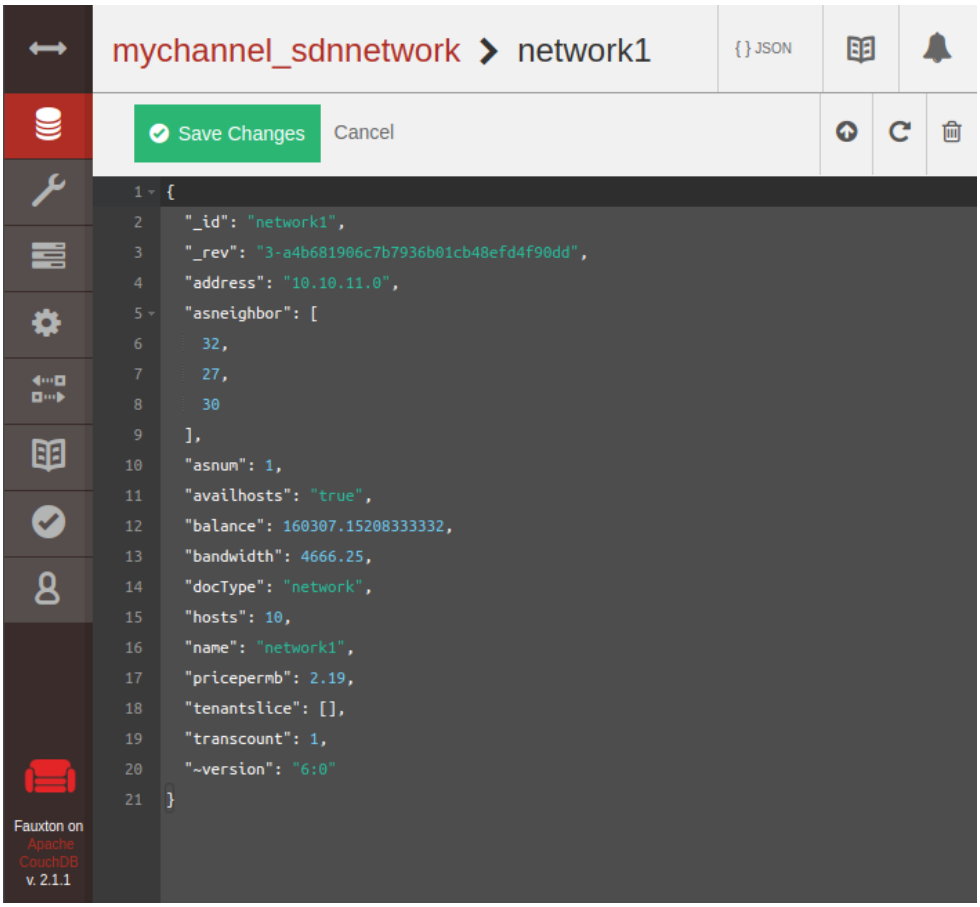


Figure 4.5. Image of the entry for network1 in the couchDB web interface.


```
],
"asnum": 1,
"availhosts": "true",
"balance": 160307.30416666664,
"bandwidth": 4656.25,
"docType": "network",
"hosts": 9,
"name": "network1",
"pricepermb": 2.19,
"tenantslice": [
  {
    "bandwidth": 10,
    "docType": "tenant",
    "duration": 25,
    "hosts": 1,
    "name": "network0",
    "saddress": [
      "10.10.10.102",
      "10.10.10.101"
```

```
      ],
      "starttime": 1539386990,
      "transnum": 2
    }
  ],
  "transcount": 2,
  "~version": "7:0"
}
```


The information in the tennantslice can now be used to install an intent in ONOS that matches the information in the transaction. Intents can be programmed to end when the duration ends, but the blockchain state must be updated after the duration has ended. The pruneTenants function can be executed after the duration has ended to remove any expired transactions from the current state.

Past transactions can be viewed by searching with the blockchain by using getHistoryForNetwork. Using getHistoryForNetwork after the invoke and one transaction returns:


```
[
  {
    "TxId": "16819e76310c5b7761ce8cc1841c903ab34879cca227f11234bf0f29c88cb0bb",
    "Value": {
      "docType": "network",
      "name": "network1",
      "address": "10.10.11.0",
      "hosts": 10,
      "availhosts": "true",
      "bandwidth": 4666.25,
      "balance": 160307,
      "pricepermb": 2.19,
      "tenantslice": null,
      "asnum": 1,
      "asneighbor": [
        32,
        27,
        30
      ],
      "transcount": 0
    },
    "Timestamp": "2018-11-01 21:58:57.68679551 +0000 UTC",
    "IsDelete": "false"
  },
  {
    "TxId": "1cc0d59d9dd525e954b6ada7c9525d1624f9d8d2800d7f88dbb5d5b69bfd4347",
    "Value": {
```

```
      "docType": "network",
      "name": "network1",
      "address": "10.10.11.0",
      "hosts": 9,
      "availhosts": "true",
      "bandwidth": 4656.25,
      "balance": 160307.15208333332,
      "pricepermb": 2.19,
      "tenantslice": [
        {
          "docType": "tenant",
          "name": "network0",
          "starttime": 1541109562,
          "duration": 25,
          "bandwidth": 10,
          "hosts": 1,
          "saddress": [
            "10.10.10.104",
            "10.10.10.103",
            "10.10.10.102",
            "10.10.10.101"
          ],
          "daddress": [
            "10.10.11.101",
            "10.10.11.102",
            "10.10.11.103",
            "10.10.11.104"
          ],
          "transnum": 1
        }
      ],
      "asnum": 1,
      "asneighbor": [
        32,
        27,
        30
      ],
      "transcount": 1
    },
    "Timestamp": "2018-11-01 21:59:22.170521677 +0000 UTC",
    "IsDelete": "false"
  }
]
```


4.3.5 Control Application

When transactions take place on the Hyperledger Fabric Network, a process needs to take the transaction and notify ONOS to implement an intent. To build a basic functioning

prototype, we will need to generate a post request in the proper format for an intent, and transmit the intent to the ONOS REST API. ONOS post requests will only be generated after a valid transaction has taken place on Hyperledger Fabric. The ideal method of implementing a notification of a valid transaction would be a push from the anchor peer to the Control Application. Developing a push notification would require modification of the Hyperledger code base or use of an SDK and will be left for future work.

Without the ability to receive push notifications, the Control Application will need to access the Hyperledger Fabric state database and pull the state of the network that it is controlling. Since the Control Application needs to pull transactions after they execute, the Control Application will be put on a loop and make pull requests repeatedly. The Control Application can gain access to the Hyperledger state via the CLI Docker container. The Control Application uses the unique transaction number transnum in the data structure in Section 4.3.3 to avoid posting duplicate intents.

The code for the Control Application is listed in Appendix A.7. The main program calls the function updateLists which calls the function queryHosts and collectHosts. queryHosts will create a sub-process that runs the readNetwork function from the chain code and returns the network state in JSON format. collectHosts generates a list of hosts currently being controlled by ONOS. A GET request to the ONOS API at the address http://<ONOS_IP>:8181/onos/v1/hosts will return a list of hosts on the network in JSON format. The hosts list is consumed into a host class object in the Control Application for easy referencing. In the main program, the tenant network's transaction number is checked for previous processing. If there is no entry to the transaction, the transaction is stored in a binary search tree (BST) with the transaction number as the key. Finally, the processIntents function will make a post request to the ONOS API at the address http://<ONOS_IP>:8181/onos/v1/intents with a JSON formatted intent; see Figure 4.4 for example intent. If the post intent request is successful, the API will return a response code 200. The ONOS intent framework will process the intent, translate it to flow rules, and install the flow rules on all appropriate switches.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Evaluation and Analysis

In this chapter, we will show a functioning prototype of an east/west protocol using Hyperledger Fabric and the Control Application. Additionally, an analysis of the Hyperledger Fabric network performance is presented.

5.1 Validation of Resource Sharing between SDNs

With the Control Application running, the transaction from Section 4.3.4, can be executed on the network shown in Figure 4.2. After execution, the Control Application pulled the transaction by running a query and installed a matching intent on the ONOS on VM 4. On VM 1, iperf was set up in client mode before the transaction and on VM 4 iperf was set up in server mode. With approximately one second of latency, the transaction processed on Hyperledger, the transaction was pulled from the Control Application, the intent posted from the control app, the intent framework processed the intent and installed flows, and the data transfer from VM 1 to VM 4 initiated, see Figure 5.1. Figure 5.2 (a) shows the data transfer on the ONOS GUI across the SDN network on VM 1 and Figure 5.2 (b) shows the same transfer on VM 4.

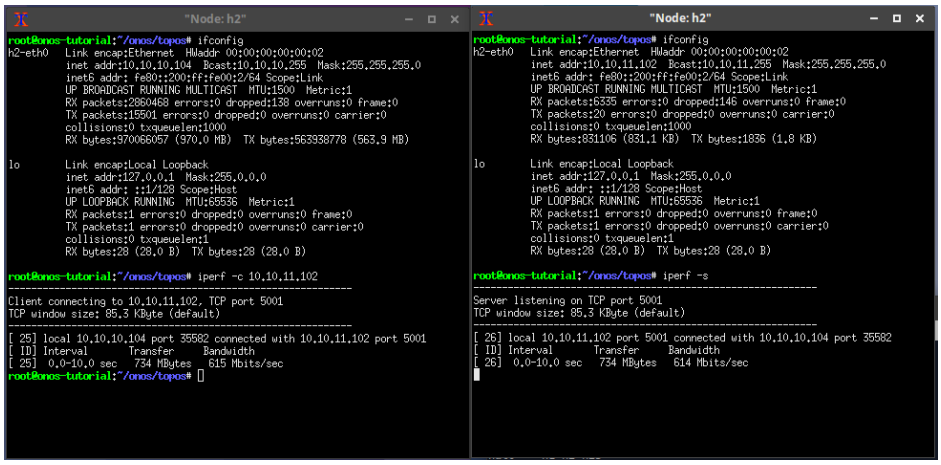
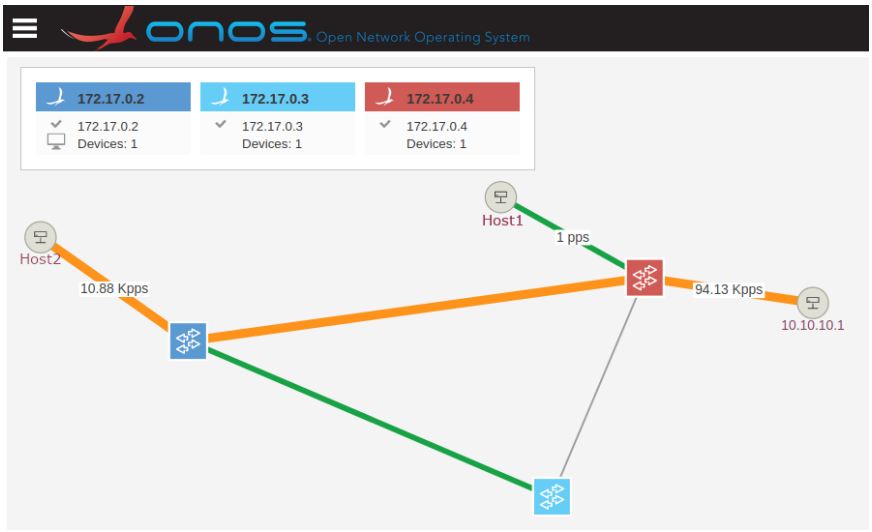
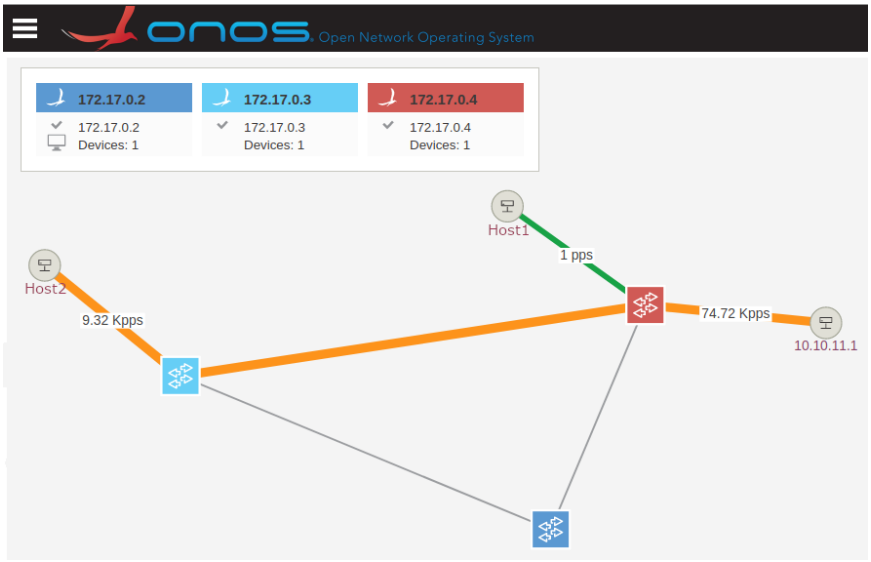


Figure 5.1. iperf transfer between VM 1 and VM 4 SDN networks. VM 1 is on the left running iperf client, and VM 4 is on the right running iperf server



(a) Screen capture of VM 1 ONOS GUI. VM 1 is the client for iperf and is network0 on the blockchain.

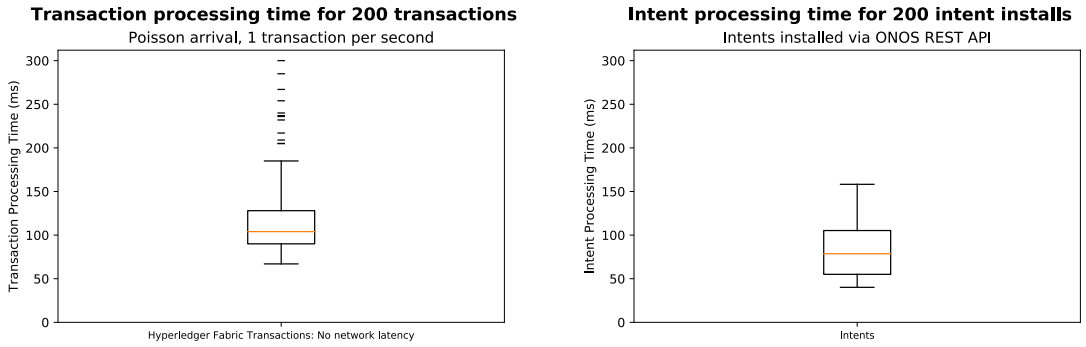


(b) Screen capture of VM 4 ONOS GUI. VM 4 is the server for iperf and is network1 on the blockchain.

Figure 5.2. Successful use of the Control Application on an SDN network after a transaction. Traffic is color-coded by bandwidth with orange representing heavy usage. The data is displayed in (kilo) packets per second. The square objects are Open vSwitch switches. The ONOS controller is running in cluster with three instances. Each switch is color-coded to the controller that is controlling it.

5.2 Performance Evaluation

Time measurements from the initiation of a transaction to its completion and the start of data transfer varied from 1s to 2s. Timing is difficult to measure and automate because timing code needs to execute on different machines. Since the most inefficient code is our prototype Control Application, data was collected for Hyperledger Fabric transactions and ONOS intent installations. Figure 5.3 (a) shows the transaction processing time for 200 transactions on Hyperledger Fabric with a Poisson arrival rate of 1 transaction per second. The mean Hyperledger Fabric transaction processing time is 116.3 ms. The data for Figure 5.3 (a) was collecting by using by timing a transaction in the CLI container. Figure 5.3 (b) shows the installation time for 200 intents to ONOS. The mean intent install time is 86.12 ms. The data for Figure 5.3 (b) was collected using the command "date +%s.%N" which prints a UNIX timestamp and nanosecond. The intent was installed using curl addressed to the ONOS REST API. Intents will be in a status "installing" until the install is completed at which time their status is "installed". Time-stamps were collected before curl was initiated and after the state changed to installed. The combined mean intent install and transaction processing is 202.42 ms. Since the data transfer took between 1s to 2s to initiate, and only approximately 202.42 is from the intent install and transaction time, the majority of the delay is attributable to inefficiencies in the Control Application. If the Control Application is optimized to receive push notifications instead of pulling in a loop, the total mean time for data transfer to initiate should be slightly above 202.42 ms.



(a) Transaction processing time. (b) ONOS Intent processing time.

Figure 5.3. Intent installation and transaction processing times.

The time between a transaction being initiated to data transferring between the test networks has two major contributing elements, the time for a transaction to process, and the time for flows to be installed on the Open vSwitches. The time for transactions to be installed is dependent on the Control Application and ONOS. Because the Control Application has not been optimized and can be redeveloped to function with any SDN controller, the time to install flows will not be evaluated and left to future work. Our analysis will focus on the performance of Hyperledger Fabric to process transactions over a network. Specifically, we will evaluate the transaction processing time for different network latencies and different transaction arrival rates.

As discussed in section 2.3.2, the significant configurable variables for Hyperledger Fabric are the block size, timeout, endorsement policy, consensus model, and choice of state database. The block size is controlled by changing the batch size with the variables, MaxMessageCount, AbsoluteMaxBytes, and PreferredMaxBytes shown in lines 138-151 of Appendix A.3.4. The timeout controls how long before a block is issued before AbsoluteMaxBytes has been issued and is controlled on line 136 of Appendix A.3.4. The endorsement policy controls which peers sign a transaction. During block validation, the endorsement policy is checked by all validating peers. The endorsement policy is controlled during instantiation of the chaincode using the $-P$ option in the instantiate or upgrade commands, see Appendix A.5.4 and Appendix A.5.5. The endorsement policy can be set to use AND, OR, and OutOf. The endorsement policy can specify an organization as well as admin, member, client or peer. As an example, the policy OutOf(2, Org1.peer, Org2.member, Org3.client) will require two of the three listed entities to endorse. Adding more entities to the endorsement policy decreases the transaction rate [12]. The choice of consensus is selected on line 129-130 of Appendix A.3.4. As of this writing, only the solo and Kafka models are available with the BFT under development.

Because the peers, orderer, MSP, and clients are operating one machine in our prototype, network latency between machines is sub-millisecond, the rate of which will depend on CPU scheduling. To simulate a more realistic network environment and measure the impact of network delays on transaction processing, tc (Traffic Control) was utilized. tc is a Linux kernel packet scheduler that can control the queuing discipline of network packets on Linux computers [40]. As an example, the interface eth0 on a Linux machine can have X ms of latency added to outbound packets using the command:

```
tc qdisc add dev eth0 root netem delay Xms
```

Because the Hyperledger Fabric nodes are running on Docker containers, we used Pumba, a network testing tool designed for Docker containers which utilize tc installed on the containers [41]. The Pumba binary can be activated for X minutes with a Y ms delay using the command:

```
./pumba_linux_amd64 netem --interface eth0 --duration X m delay --time Y <container name>
```

The focus of the analysis here is on the impact of network latency to transaction processing time. The work of Thakkar et al. was primarily focused on optimizing Hyperledger Fabric for transaction volume [12]. Our prototype for the east/west interface can only install flows once a transaction has processed so, the transaction processing time will be the most important metric. There are possible use cases for an east/west interface could be interested in higher transaction volumes but, Thakkar et al. shows that the rates can be increased by adding additional channels [12].

For the testing conducted, the default values of the configurable variables were kept including the JSON formatted data structure with CouchDB. By keeping the values static, we can better isolate impacts to transaction processing time as network latency is varied. However, the results will not show the optimal processing time of which the system is capable. To test, we built a Hyperledger Fabric network with three organizations. Each organization will have two peers with only one of the two being anchor peers. We generated 200 transactions using a Poisson arrival process with an arrival rate of 1, 5, and 10 transactions per second. The choice of these transaction rates is to represent a resource sharing scenario between data centers. Such a scenario typically involves large workloads, flows are generally long-lived, and as a result, transaction rates are relatively small. The start time and end time of the transactions were recorded.

Figure 5.4 shows no delay, 10 ms of delay on the orderer, 10 ms of delay on the anchor peer for organization 1 (p1), 10 ms delay on p1 and the orderer, 10 ms of delay on p1, p2, and

the orderer, and 10 ms of delay on p1, p2, p3, and the orderer. All data in Figure 5.4 use the same Poisson arrival values. The endorsing peer for all transactions is p1. Adding 10 ms of latency reveals the following results:

- For 1 transaction per second, 10 ms to the orderer added 68.2 ms of latency to the mean.
- For 1 transaction per second, 10 ms of latency to p1 added 175.17 ms to the mean.
- For 1 transaction per second, 10 ms of latency to both p1 and the orderer adds 229.6 ms of latency which is slightly under 243.2 ms, the sum of latency from p1 and the orderer alone.
- For 1 transaction per second, 10 ms of latency on non-endorsing peers p2 and P3 do not impact the transaction processing time.
- Delay on p1 is more impactful than the delay on the orderer for transaction rates of 5 and 10 transactions per second.
- Delay on non-endorsing peers does not impact transaction processing time at 5 and 10 transactions per second.
- 10 ms of delay on the orderer increases transaction processing time as the transaction rate increases from 181.86 ms to 219.19, 217.295 ms for 5 and 10 transactions per second respectively.
- 10 ms of delay on p1 increases transaction processing time as the transaction rate increases from 288.82 ms to 381.81, 412.38 ms for 5 and 10 transactions per second respectively.

Figure 5.5 shows the same test as Figure 5.4 but at 20 ms of latency. Adding 20 ms of latency reveals the following results:

- For 1 transaction per second, 20 ms to the orderer added 109.7 ms of latency to the mean.
- For 1 transaction per second, 20 ms of latency to p1 added 329.86 ms to the mean.
- For 1 transaction per second, 20 ms of latency to both p1 and the orderer is adds 437.835 ms of latency which is comparable to 439.56 ms, the sum of latency from p1 and the orderer alone.
- For 1 transaction per second, 20 ms of latency on non-endorsing peers p2 and P3 do not impact the transaction processing time.

- Delay on p1 is more impactful than the delay on the orderer for transaction rates of 5 and 10 transactions per second.
- Delay on non-endorsing peers does not impact transaction processing time at 5 and 10 transactions per second.
- For 20 ms of delay on the orderer, the transaction processing time increases as the transaction rate increases from 229.32ms to 290.41, 286.44 ms for 5 and 10 transactions per second respectively.
- 10 ms of delay on p1 increases transaction processing time as the transaction rate increases from 449.435 ms to 594.51, 640.62 ms for 5 and 10 transactions per second respectively.

Comparing both sets of data in Figure 5.4 and Figure 5.5, the transaction processing time slightly decreased between 5 and 10 transactions per second for both 10 and 20 ms of delay on the orderer. The decrease can be explained by the observation of Thakkar et al. that the transaction latency increased above and below the saturation point [12]. The transaction processing time was strictly increasing between 5 and 10 transactions per second for both 10 and 20 ms of delay. After reviewing the container logs for 1, 5 and 10 transactions per second, we see an increase in multiple transactions per block.

At 1 transaction per second, a sample of p1's log shows:

```
Channel [sdnnetwork]: Committed block [1] with 1 transaction(s)
Channel [sdnnetwork]: Committed block [2] with 1 transaction(s)
Channel [sdnnetwork]: Committed block [2] with 1 transaction(s)
Channel [sdnnetwork]: Committed block [3] with 1 transaction(s)
Channel [sdnnetwork]: Committed block [5] with 1 transaction(s)
Channel [sdnnetwork]: Committed block [6] with 1 transaction(s)
Channel [sdnnetwork]: Committed block [7] with 1 transaction(s)
...
```

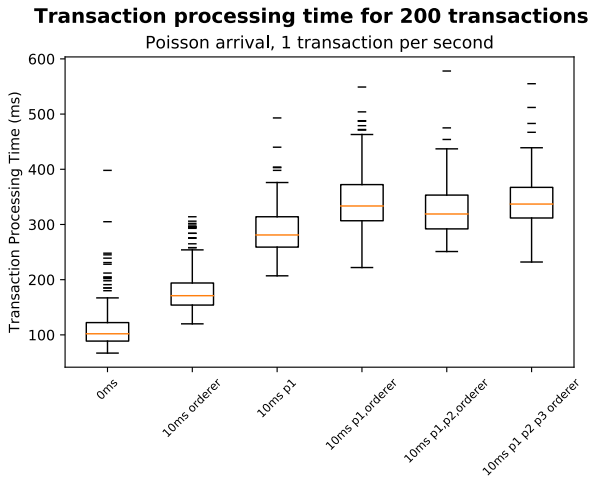
At 5 transactions per second, a sample of p1's log shows:

```
Channel [sdnnetwork]: Committed block [1] with 3 transaction(s)
Channel [sdnnetwork]: Committed block [2] with 3 transaction(s)
Channel [sdnnetwork]: Committed block [3] with 3 transaction(s)
Channel [sdnnetwork]: Committed block [4] with 3 transaction(s)
Channel [sdnnetwork]: Committed block [5] with 2 transaction(s)
```

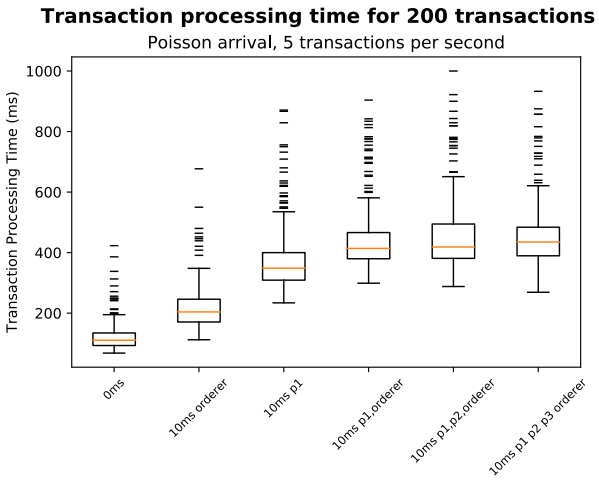
```
Channel [sdnnetwork]: Committed block [6] with 2 transaction(s)
Channel [sdnnetwork]: Committed block [7] with 2 transaction(s)
...
```

Combining transactions in one block will decrease the overall impact of delay on the orderer. However, delay on p1 had a higher impact overall. Increasing the number of blocks (1 per transaction) increases the overall number of blocks needing signature verification by the MSP of the endorsing peer. Thakkar et al. introduced a cache of deserialized identities which reduced cryptographic overhead and prevented a RTT for the verification of each block significantly improving transaction rates [12]. Introducing a deserialized identity cache will likely significantly improve transaction processing time especially as the network latency increases.

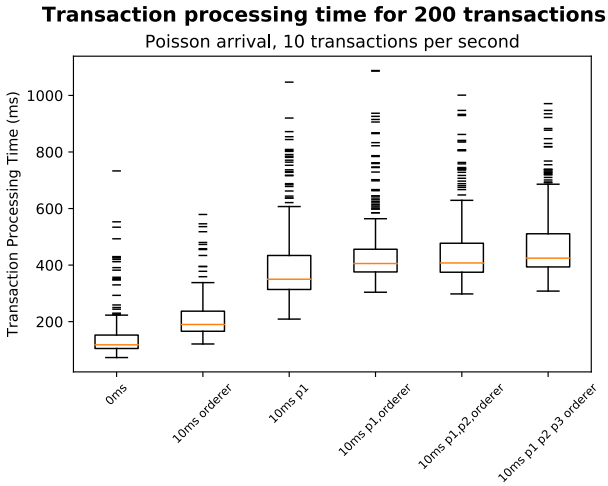
To better see how network latency impacts the transaction processing time, different delays were used on the orderer, p1, and p3 individually and shown in Figure 5.6, Figure 5.7, and Figure 5.8 respectively. Regression analysis shows that transaction processing time is linear with respect to network latency with excellent fit. As an example, regression analysis is shown for Figure 5.6 (a) in Figure 5.9. Therefore, we can conclude that transaction processing time increases linearly as network latency increases.



(a) 1 transaction per second expected arrival.

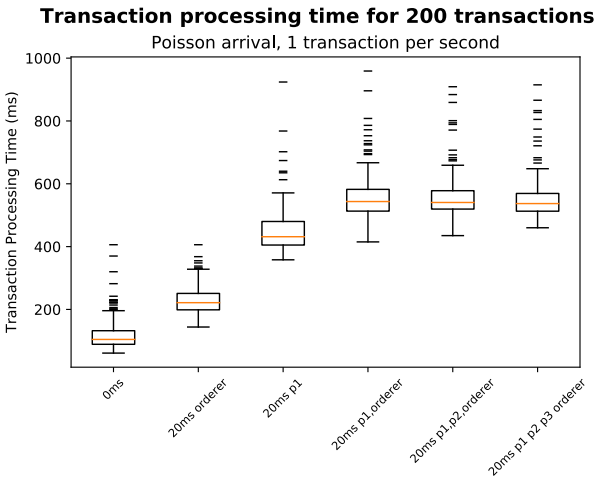


(b) 5 transactions per second expected arrival.

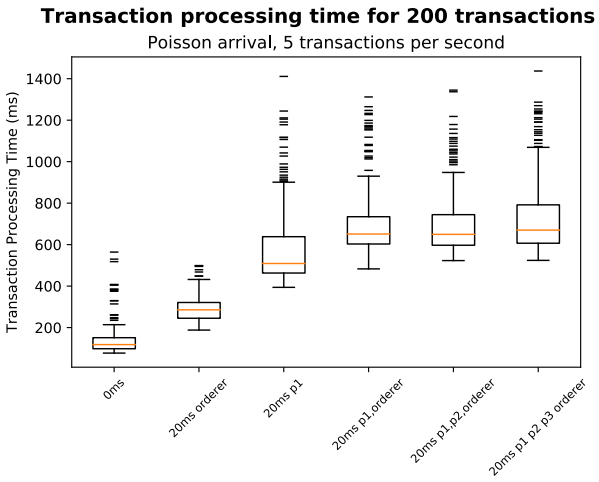


(c) 10 transactions per second expected arrival.

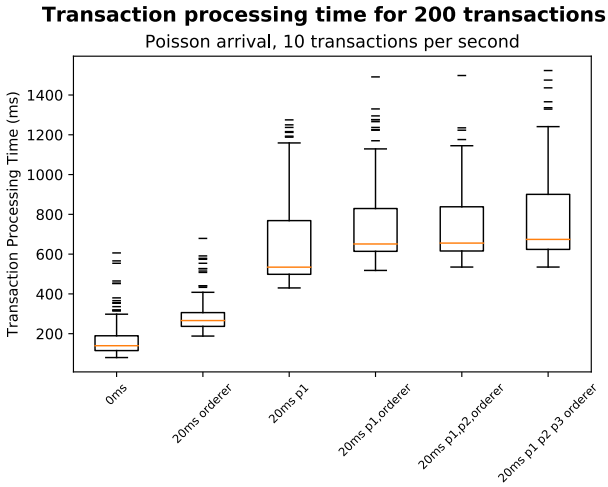
Figure 5.4. A Combination of Hyperledger Fabric entities with 10 ms latency for 1, 5, 10 transactions per second expected arrival rates.



(a) 1 transaction per second expected arrival.

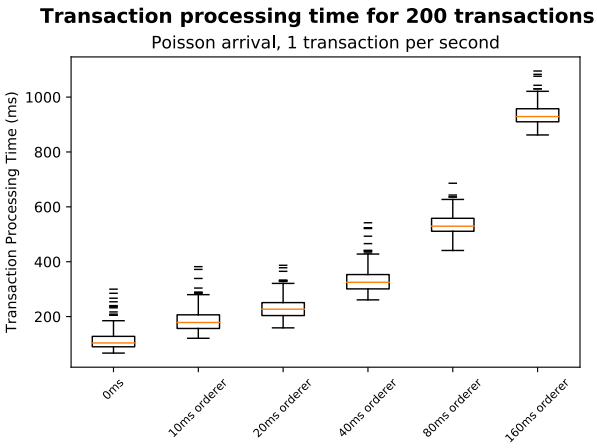


(b) 5 transactions per second expected arrival.

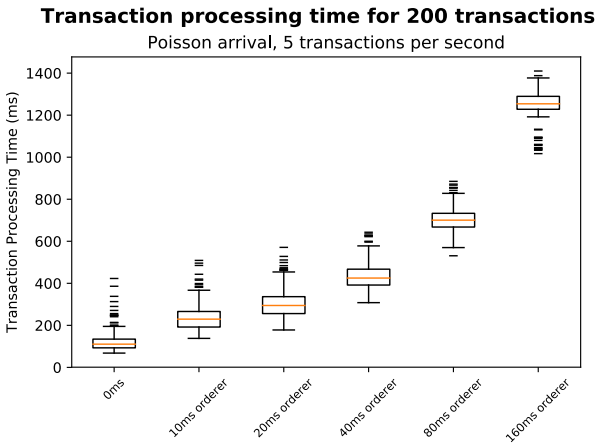


(c) 10 transactions per second expected arrival.

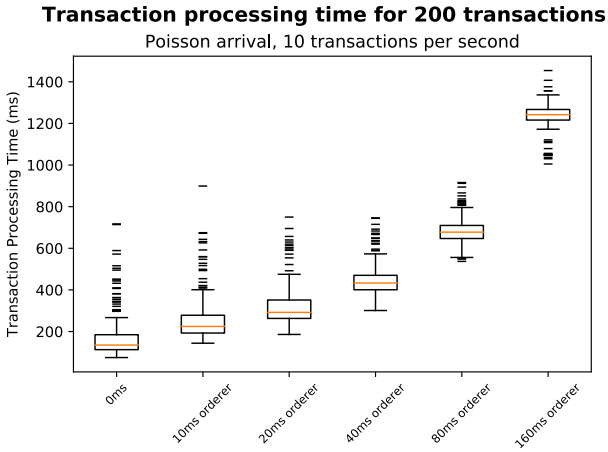
Figure 5.5. A Combination of Hyperledger Fabric entities with 20 ms of network latency for 1, 5, 10 transactions per second expected arrival rates.



(a) 1 transaction per second expected arrival.

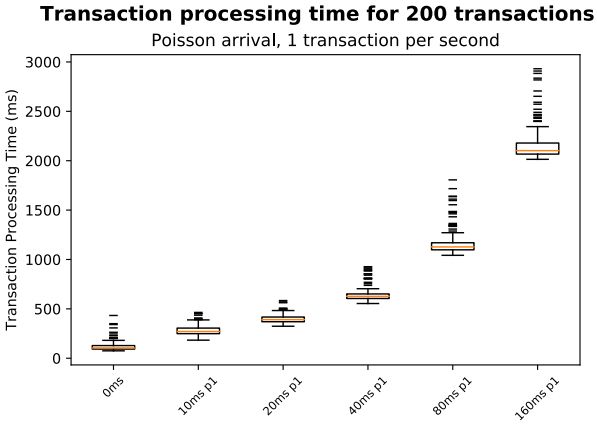


(b) 5 transactions per second expected arrival.

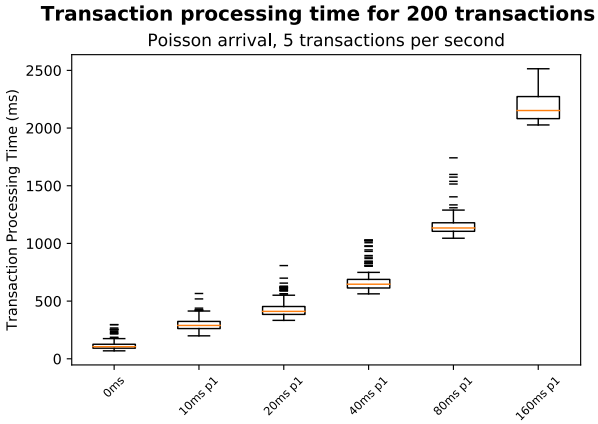


(c) 10 transactions per second expected arrival.

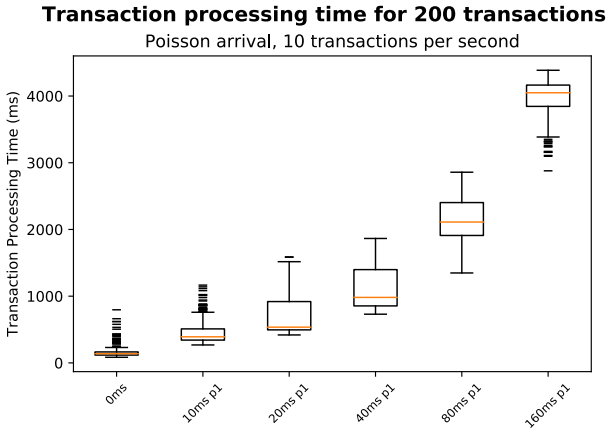
Figure 5.6. Hyperledger Fabric orderer with 10, 20, 40, 80, and 160 ms of network latency for 1, 5, 10 transactions per second expected arrival rate.



(a) 1 transaction per second expected arrival.

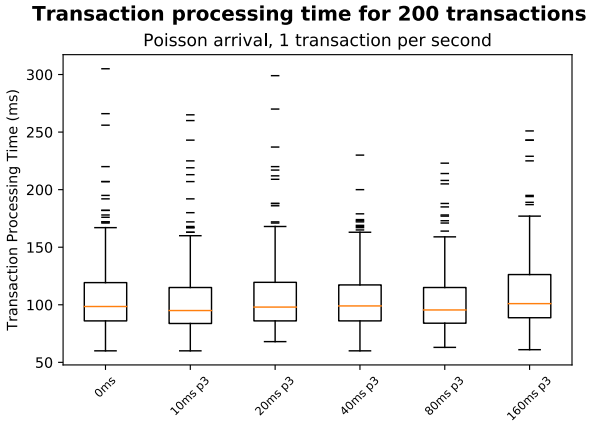


(b) 5 transactions per second expected arrival.

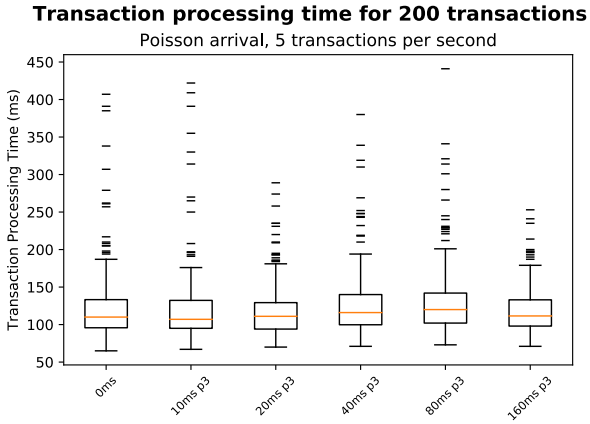


(c) 10 transactions per second expected arrival.

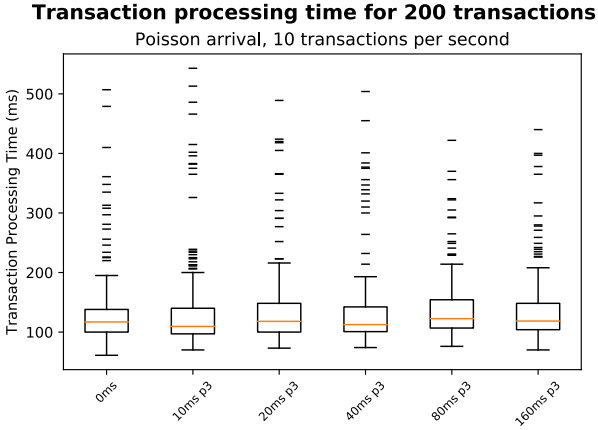
Figure 5.7. Hyperledger Fabric endorsing peer with 10, 20, 40, 80, and 160 ms of network latency for 1, 5, 10 transactions per second expected arrival rate.



(a) 1 transaction per second expected arrival.

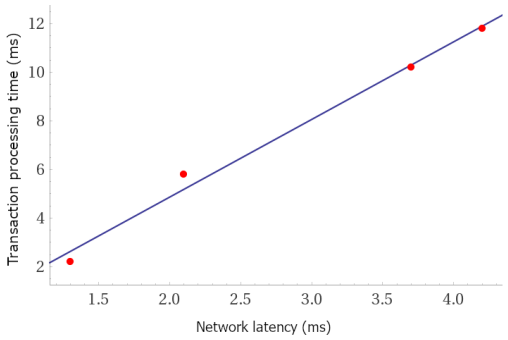


(b) 5 transactions per second expected arrival.

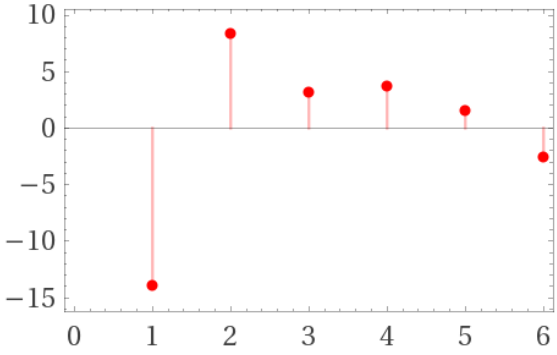


(c) 10 transactions per second expected arrival.

Figure 5.8. Hyperledger Fabric non-endorsing peer with 10, 20, 40, 80, and 160 ms of network latency for 1, 5, 10 transactions per second expected arrival rate.



(a) Linear fit.



(b) Residuals.

AIC	BIC	R^2	adjusted R^2
46.8685	46.2437	0.999358	0.999197

(c) Fit data.

Figure 5.9. Regression analysis of Figure 5.6 (a) with a best-fit curve $130.005 + 5.05452x$.

CHAPTER 6:
Conclusions and Future Work

In this chapter, we present our conclusions from our implementation and evaluation and analysis. We conclude the chapter with a discussion on future work.

6.1 Conclusions

In this research, we were able to answer our primary research question of building an SDN east/west protocol using a Control Application and a blockchain protocol, Hyperledger Fabric. Our prototype is capable of linking separately managed SDN networks, which currently available methods are incapable of doing with assured verifiability. The properties of permissioned blockchains, immutability, serializability, verifiability, and chaincode, provide a level of guarantees for security, autonomy, and privacy. For example, Le et al. point out that determining the impacts of multiple flow requests remains a challenge for the development of SFP [10]. By serializing the flow requests, our prototype can be extended to check metric information before executing, making multi-flow impacts deterministic.

In Section 5.2, the impact of network latency on the orderer, endorsing peers, and non-endorsing peers to the transaction processing time was shown. An increase in latency caused an increase in transaction processing time with a linear relationship. The transaction processing times on our test network for 10 ms and 20 ms of latency are approximately 400 ms and 600 ms respectively and increase slightly as the transaction rates increase. The Hyperledger Fabric transaction processing times are significantly faster than Ethereum and Bitcoin, but the demonstrated transaction times may not be fast enough to be a replacement for BGP in SDN networks as Le et al. envisioned SFP to be. However, this research did not optimize Hyperledger Fabric for transaction processing time and transaction volume so the minimum values are unknown and will vary from network to network. For some long-lived flows such as data center connectivity, the demonstrated transaction times would be sufficient to connect separately managed SDN networks. Interestingly, transaction processing time and transaction volume can be improved at the cost of CPU utilization by increasing the number of channels [12]. Since this research shows that network latency plays an important role in transaction processing times, Hyperledger Fabric nodes should

be carefully positioned to improve scalability.

The Hyperledger state database allows network entities to share metric data. Transactions update the accounts state with available bandwidth, hosts, and other metrics. Because each account controls the data they share and the transactions they accept, the autonomy of each network is maintained. Chaincode can prevent the modification of account state data unless approved by the account owner.

Le et al. state that subscription requests should be responded to in a reasonable amount of time [10]. In our prototype, subscription requests correspond to joining a Hyperledger Fabric channel and publications are transactions. Creating a channel between peers is a fast operation. Joining an existing channel requires permission from the channel members and a transfer of chaincode and prior block data. Permission granting operations are fast, but the transfer of a large amount of prior block data is not. Alternatively, a new blockchain can be created for including the new member to increase the speed of joining a channel. The drawback to this is historical transaction data will not be linked to the new chain and not available to the new member, but the original members can retain and search the old chain. Creating a new blockchain is most useful in the case when the new subscriber wants to only link to a subset of the existing subscribers.

The system can maintain the privacy of the information published by one network because the information is only shared to networks on a channel and the network controls what it publishes in its state. When a network agrees to route another network's data, internal network information does not need to be shared. If a network requests to connect to a service, flow modifications can be used to hide internal network information. The requesting network will send packets to the supporting networks published IP address and the supporting networks internal flows can direct the packets to the service. Le et al. mention that multiple flow requests might reveal internal network information [10]. For example, if a network requests to connect at a specific IP address, the supporting network can agree to the transaction and install flows internally. By agreeing, the supporting network is implicitly stating that it can connect to the requested IP. If the requesting network makes multiple requests, the aggregate set of requests can reveal the internal network map of the supporting network. Since each transaction request is saved in the blockchain, the blockchain serves as a record of the scan attempt. Since the Hyperledger Fabric is a permissioned blockchain,

user identity can be a requirement to join the network so the identity of the offender would be known. Additionally, chaincode can be used enforce financial penalties, or the offending account can be banned from the channel.

6.2 Future Work

Development of a proof-of-concept showed that an east/west interface is possible using blockchains. However, before the system is shown to be fully viable, additional work is needed. In this section covers suggestions for future work to further develop the concepts in this thesis.

Thakkar et al. made optimizations to Hyperledger Fabric to improve the transaction volume [12]. Similar work needs to be performed to find a balance between transaction processing time and transaction volume. Some optimizations discovered by Thakkar et al. should be implemented and fine-tuned to find a balance between transaction processing time and transaction volume. Most importantly, the implementation of deserialized identity caching which already significantly increases transaction volume would also reduce transaction processing time because it saves an RTT for requesting a check, and the CPU costs of repeated cryptographic verification. To properly perform this analysis the work should be conducted using a hardware setup to remove the bottlenecks and artificialities of vitalization.

Le et al. require SFP to be autonomous and stable because SFP was envisioned a protocol to replace BGP as a routing protocol. The research here did not create the prototype that can be used to find routes between SDN networks. However, the system we built can be extended to allow routing between networks by making adjustments to the chaincode and data structures. We will discuss two possible methods that could allow routing between SDN network to function. The first is to create a channel where members publish neighbor information. A path search can determine the SDN networks needed to reach a destination and flows can be requested from those networks via private channels between them. Stable algorithms can be employed to find paths. The major problem with the approach is that network outages would not be reflected in the state database and would require a transaction from a neighbor to update neighbors. The second method is to use BGP or any other routing protocol as a default routing system and modify flows from default when necessary. Using BGP and adding flows to modify the default behavior is how SDX, an SDN east/west protocol,

functions [14]. The proof-of-concept in this research should be developed to function in the same scenario as SDX and comparisons should be made between the protocols.

The work in this project used the solo orderer and a Control Application written in python. The next version of the prototype should use the Hyperledger Fabric SDK in the development of the Control Application and should use push notifications when transactions complete. BFT consensus algorithm was not available on Hyperledger Fabric during this project. When BFT is available, it should be tested and compared to Kafka for transaction volume and transaction processing time.

The development of a key-value structure that allows for similar functionality to the JSON data structure used in this work will allow comparison of the Go LevelDB and CouchDB and evaluate them for transaction processing time. Additionally, the JSON data structure used here should be optimized to improve efficiency. Optimizing the data structure can also reduce the size of blocks reducing storage and network resources.

In this research, we did not adequately determine if the proof-of-concept is scalable. Acceptable trade-offs between transaction processing time and transaction volume need to be determined. For example, our use of JSON and the couchDB to demonstrate resource sharing decreased transaction volume, increased transaction processing time, and increased the size of blocks when compared to using simple data structures and the Go LevelDB. Ultimately, with the data structure we used, we will likely not be able to reach the transaction volumes of 3500 transactions per second by Androulaki et al. [29] or the 2700 transactions per second by Thakkar et al. [12] if the tests were run on the same equipment. The minimum amount of resource sharing, acceptable transaction volume, and acceptable transaction processing time can only be defined for a specific use case. Therefore, the proof-of-concept used here should be developed and optimized for a specific use to determine its acceptability. Additionally, we did not perform tests on the storage size of the blockchain over time. The storage size will depend on many factors including the optimization of the resource sharing and the expected arrival rates which will ultimately depend on the use case.

Finally, there is an opportunity to implement network security using the concepts developed in this work. Buterin proposed using Ethereum as an identity and reputation system [26]. A similar system could be built using Hyperledger Fabric and combined with the system developed in this work. Users on a network would register flows to their identity, and an

SDN controller can check flow requests against an access list before installing the flows. Checking flows against a registered identity could provide network layer access to sensitive computers and servers.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: Network Topology, YAML Configuration, Scripts, Contract Code

A.1 Mininet Example

A custom topology for building a Mininet network with three hosts and three switches. Code adapted from [19].

```
from mininet.net import Mininet
from mininet.node import Controller, OVSSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def remoteControllerNet():

    net = Mininet( controller=RemoteController, switch=OVSSwitch)

    info( "Creating remote controller\n" )
    c1 = RemoteController( 'c1', ip='172.17.0.2', port=6633, protocols=["OpenFlow13"]  )
    info( "Creating switches\n" )
    #Add 3 switches using OpenFlow v1.3
    s1 = net.addSwitch( 's1', protocols=["OpenFlow13"] )
    s2 = net.addSwitch( 's2',protocols=["OpenFlow13"] )
    s3 = net.addSwitch( 's3',protocols=["OpenFlow13"] )

    info( "Creating hosts\n" )
    hostsSwitch1 = [ net.addHost( 'h%d' % n ) for n in [1, 2] ] #add hosts named h1,h2
    hostsSwitch2 = [ net.addHost( 'h%d' % n ) for n in [3, 4] ] #add hosts named h3,h4
    hostsSwitch3 = [ net.addHost( 'h%d' % n ) for n in [5, 6] ] #add hosts named h5,h6

    info( "Creating links\n" )
    for h in hostsSwitch1:
        net.addLink( s1, h )
    for h in hostsSwitch2:
        net.addLink( s2, h )
    for h in hostsSwitch3:
        net.addLink( s3, h )
    net.addLink( s1, s2 )
    net.addLink( s2, s3 )
    net.addLink( s3, s1 )

    info( "Starting network\n" )
    net.build()
    s1.start( [ c1 ] ) #Connect switch 1 to remote controller.
```

```
    s2.start( [ c1 ] ) #Connect switch 1 to remote controller.
    s3.start( [ c1 ] ) #Connect switch 1 to remote controller.

    info( "Running CLI\n" )
    CLI( net )

if __name__ == '__main__':
    setLogLevel( 'info' )  # output info lines on CLI
    remoteControllerNet()
```

A.2 Network Topology

Mininet topology for building the network in Figure 4.2. The code is a modified version of default.py from [42].

```
#!/usr/bin/env python

from mininet.topo import Topo

class AttMplsTopo( Topo ):

    def addSwitch( self, name, **opts ):
        kwargs = { 'protocols' : 'OpenFlow13' }  #Selects OpenFlow 1.3 as the default protocol.
        kwargs.update( opts )
        return super(AttMplsTopo, self).addSwitch( name, **kwargs )

    def __init__( self ):

        # Initialize Topology
        Topo.__init__( self )

        # Create switches
        Core1 = self.addSwitch( 's1' )
        Core2 = self.addSwitch( 's2' )
        Core3 = self.addSwitch( 's3' )

        # Create hosts
        Core1_host1 = self.addHost( 'h1' )
        Core2_host2 = self.addHost( 'h2' )
        Core3_host3 = self.addHost( 'h3' )


        # Create links between switches and hosts.
        self.addLink( Core3 , Core3_host3 )
        self.addLink( Core1 , Core1_host1 )
        self.addLink( Core2 , Core2_host2 )
```

```
32
33
34          # Create links between hosts.
35          self.addLink( Core3 , Core1)
36          self.addLink( Core3 , Core2)
37          self.addLink( Core2 , Core1)
38
39
40  topos = { 'att': ( lambda: AttMplsTopo() ) }
41
42  if __name__ == '__main__':
43      from onosnet import run
44      run( AttMplsTopo() )
```

The program default.py imports the function "run" from onosnet.py. Run is used to connect the network topology laid out in default.py to a cluster of ONOS controllers. Because the code uses onosnet.py it will require ONOS to be installed or be modified to run with the default Mininet controller. The code for onosnet.py is presented here unmodified from its source [43].

```
1   #!/usr/bin/python
2   import itertools
3   import os
4   import signal
5   import sys
6   from argparse import ArgumentParser
7   from subprocess import call
8   from threading import Thread
9   from time import sleep
10
11  import gratuitousArp
12  from mininet.cli import CLI
13  from mininet.examples.controlnet import MininetFacade
14  from mininet.link import TCLink
15  from mininet.log import info, output, error
16  from mininet.log import setLogLevel
17  from mininet.net import Mininet
18  from mininet.node import RemoteController, Node
19
20  ARP_PATH = gratuitousArp.__file__.replace('.pyc', '.py')
21
22  class ONOSMininet( Mininet ):
23
24      def __init__( self, controllers=[], gratuitousArp=True, build=True, *args, **kwargs ):
25          """Create Mininet object for ONOS.
26          controllers: List of controller IP addresses
```

```
27          gratuitousArp: Send an ARP from each host to aid controller's host discovery"""
28
29          # delay building for a second
30          kwargs[ 'build' ] = False
31
32          Mininet.__init__(self, *args, **kwargs )
33
34          self.gratArp = gratuitousArp
35
36          # If a controller is not provided, use list of remote controller IPs instead.
37          if 'controller' not in kwargs or not kwargs['controller']:
38              info ( '*** Adding controllers\n' )
39              ctrl_count = 0
40              for controllerIP in controllers:
41                  self.addController( 'c%d' % ctrl_count, RemoteController, ip=controllerIP )
42                  info( '   c%d (%s)\n' % ( ctrl_count, controllerIP ) )
43                  ctrl_count = ctrl_count + 1
44
45          if self.topo and build:
46              self.build()
47
48      def start( self ):
49          Mininet.start( self )
50          if self.gratArp:
51              self.waitConnected( timeout=5 )
52              sleep(2)
53              info ( '*** Sending a gratuitious ARP from each host\n' )
54              self.gratuitousArp()
55
56      def verifyHosts( self, hosts ):
57          for i in range( len( hosts ) ):
58              if isinstance( hosts[i], str):
59                  if hosts[i] in self:
60                      hosts[i] = self[ hosts[i] ]
61                  else:
62                      info( '*** ERROR: %s is not a host\n' % hosts[i] )
63                      del hosts[i]
64              elif not isinstance( hosts[i], Node):
65                  del hosts[i]
66
67      def gratuitousArp( self, hosts=[] ):
68          "Send an ARP from each host to aid controller's host discovery; fallback to ping if necessary"
69          if not hosts:
70              hosts = self.hosts
71          self.verifyHosts( hosts )
72
73          for host in hosts:
74              info( '%s ' % host.name )
75              info( host.cmd( ARP_PATH ) )
76          info ( '\n' )
77
78      def pingloop( self ):
```

```
79          "Loop forever pinging the full mesh of hosts"
80          setLogLevel( 'error' )
81          try:
82              while True:
83                  self.ping()
84          finally:
85              setLogLevel( 'info' )
86
87      def bgIperf( self, hosts=[], seconds=10 ):
88          self.verifyHosts( hosts )
89          servers = [ host.popen("iperf -s") for host in hosts ]
90
91          clients = []
92          for s, d in itertools.combinations(hosts, 2):
93              info ( '%s <--> %s\n' % ( s.name, d.name ))
94              cmd = 'iperf -c %s -t %s -y csv' % (d.IP(), seconds)
95              p = s.popen(cmd)
96              p.s = s.name
97              p.d = d.name
98              clients.append(p)
99
100         def handler (_signum, _frame):
101             raise BackgroundException()
102         oldSignal = signal.getsignal(signal.SIGTSTP)
103         signal.signal(signal.SIGTSTP, handler)
104
105         def finish( verbose=True ):
106             for c in clients:
107                 out, err = c.communicate()
108                 if verbose:
109                     if err:
110                         info( err )
111                     else:
112                         bw = out.split( ',' )[8]
113                         info( '%s <--> %s: %s\n' % ( c.s, c.d, formatBw(bw) ) )
114             for s in servers:
115                 s.terminate()
116
117         try:
118             info ( 'Press ^Z to continue in background or ^C to abort\n')
119             progress( seconds )
120             finish()
121         except KeyboardInterrupt:
122             for c in clients:
123                 c.terminate()
124             for s in servers:
125                 s.terminate()
126         except BackgroundException:
127             info( '\n*** Continuing in background...\n' )
128             t = Thread( target=finish, args=[ False ] )
129             t.start()
130         finally:
```

```
131             #Disable custom background signal
132             signal.signal(signal.SIGTSTP, oldSignal)
133
134 def progress(t):
135     while t > 0:
136         sys.stdout.write( '.' )
137         t -= 1
138         sys.stdout.flush()
139         sleep(1)
140     print
141
142 def formatBw( bw ):
143     bw = float(bw)
144     if bw > 1000:
145         bw /= 1000
146         if bw > 1000:
147             bw /= 1000
148             if bw > 1000:
149                 bw /= 1000
150                 return '%.2f Gbps' % bw
151             return '%.2f Mbps' % bw
152         return '%.2f Kbps' % bw
153     return '%.2f bps' % bw
154
155 class BackgroundException( Exception ):
156     pass
157
158
159 def get_mn(mn):
160     if isinstance(mn, ONOSMininet):
161         return mn
162     elif isinstance(mn, MininetFacade):
163         # There's more Mininet objects instantiated (e.g. one for the control network in onos.py).
164         for net in mn.nets:
165             if isinstance(net, ONOSMininet):
166                 return net
167     return None
168
169
170 def do_bgIperf( self, line ):
171     args = line.split()
172     if not args:
173         output( 'Provide a list of hosts.\n' )
174
175     #Try to parse the '-t' argument as the number of seconds
176     seconds = 10
177     for i, arg in enumerate(args):
178         if arg == '-t':
179             if i + 1 < len(args):
180                 try:
181                     seconds = int(args[i + 1])
182                 except ValueError:
```

```
183                     error( 'Could not parse number of seconds: %s', args[i+1] )
184                 del(args[i+1])
185             del args[i]
186
187     hosts = []
188     err = False
189     for arg in args:
190         if arg not in self.mn:
191             err = True
192             error( "node '%s' not in network\n" % arg )
193         else:
194             hosts.append( self.mn[ arg ] )
195     mn = get_mn( self.mn )
196     if "bgIperf" in dir( mn ) and not err:
197         mn.bgIperf( hosts, seconds=seconds )
198     else:
199         output('Background Iperf is not supported.\n')
200
201 def do_gratuitousArp( self, line ):
202     args = line.split()
203     mn = get_mn(self.mn)
204     if "gratuitousArp" in dir( mn ):
205         mn.gratuitousArp( args )
206     else:
207         output( 'Gratuitous ARP is not supported.\n' )
208
209 CLI.do_bgIperf = do_bgIperf
210 CLI.do_gratuitousArp = do_gratuitousArp
211
212 def parse_args():
213     parser = ArgumentParser(description='ONOS Mininet')
214     parser.add_argument('--cluster-size', help='Starts an ONOS cluster with the given number of
      ↪   instances',
215                         type=int, action='store', dest='clusterSize', required=False, default=0)
216     parser.add_argument('--netcfg', help='Relative path of the JSON file to be used with netcfg',
217                         type=str, action='store', dest='netcfgJson', required=False, default='')
218     parser.add_argument('ipAddrs', metavar='IP', type=str, nargs='*',
219                         help='List of controller IP addresses', default=[])
220     return parser.parse_args()
221
222 def run( topo, controllers=None, link=TCLink, autoSetMacs=True):
223     if not topo:
224         print 'Need to provide a topology'
225         exit(1)
226
227     args = parse_args()
228
229     if not controllers and len(args.ipAddrs) > 0:
230         controllers = args.ipAddrs
231
232     if not controllers and args.clusterSize < 1:
233         print 'Need to provide a list of controller IPs, or define a cluster size.'
```

```
234         exit( 1 )
235
236     setLogLevel( 'info' )
237
238     if args.clusterSize > 0:
239         if 'ONOS_ROOT' not in os.environ:
240             print "Environment var $ONOS_ROOT not set (needed to import onos.py)"
241             exit( 1 )
242         sys.path.append(os.environ["ONOS_ROOT"] + "/tools/dev/mininet")
243         from onos import ONOSCluster, ONOSOVSSwitch, ONOSCLI
244         controller = ONOSCluster('c0', args.clusterSize)
245         onosAddr = controller.nodes()[0].IP()
246         net = ONOSMininet( topo=topo, controller=controller, switch=ONOSOVSSwitch, link=link,
247                            autoSetMacs=autoSetMacs )
248         cli = ONOSCLI
249     else:
250         onosAddr = controllers[0]
251         net = ONOSMininet(topo=topo, controllers=controllers, link=link, autoSetMacs=autoSetMacs)
252         cli = CLI
253     net.addNAT().configDefault()
254
255     #net.addBridge().configDefault()
256     net.start()
257
258     if len(args.netcfgJson) > 0:
259         if not os.path.isfile(args.netcfgJson):
260             error('*** WARNING no such netcfg file: %s\n' % args.netcfgJson)
261         else:
262             info('*** Setting netcfg: %s\n' % args.netcfgJson)
263             call(("onos-netcfg", onosAddr, args.netcfgJson))
264
265     cli( net )
266     net.stop()
```

A.3 Hyperledger Network Configuration YAML

A.3.1 crypto-config.yaml

crypto-config.yaml [36] is included in the first-network folder in the Hyperledger example files. It is used to define the participants in the Hyperledger network so that Public Key Infrastructure (PKI) keys are generated when the byfn.sh script is executed. The code is presented here unmodified from the source.

```
1 # Copyright IBM Corp. All Rights Reserved.
2 #
```

```
# SPDX-License-Identifier: Apache-2.0
#

# ---------------------------------------------------------------------------
# "OrdererOrgs" - Definition of organizations managing orderer nodes
# ---------------------------------------------------------------------------
OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
    Domain: example.com
    # ---------------------------------------------------------------------------
    # "Specs" - See PeerOrgs below for complete description
    # ---------------------------------------------------------------------------
    Specs:
      - Hostname: orderer
# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# ---------------------------------------------------------------------------
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org1
  # ---------------------------------------------------------------------------
  - Name: Org1
    Domain: org1.example.com
    EnableNodeOUs: true
    # ---------------------------------------------------------------------------
    # "Specs"
    # ---------------------------------------------------------------------------
    # Uncomment this section to enable the explicit definition of hosts in your
    # configuration.  Most users will want to use Template, below
    #
    # Specs is an array of Spec entries.  Each Spec entry consists of two fields:
    #   - Hostname:   (Required) The desired hostname, sans the domain.
    #   - CommonName: (Optional) Specifies the template or explicit override for
    #                 the CN.  By default, this is the template:
    #
    #                              "{{.Hostname}}.{{.Domain}}"
    #
    #                 which obtains its values from the Spec.Hostname and
    #                 Org.Domain, respectively.
    # ---------------------------------------------------------------------------
    # Specs:
    #   - Hostname: foo # implicitly "foo.org1.example.com"
    #     CommonName: foo27.org5.example.com # overrides Hostname-based FQDN set above
    #   - Hostname: bar
    #   - Hostname: baz
    # ---------------------------------------------------------------------------
    # "Template"
    # ---------------------------------------------------------------------------
    # Allows for the definition of 1 or more hosts that are created sequentially
```

```
55      # from a template. By default, this looks like "peer%d" from 0 to Count-1.
56      # You may override the number of nodes (Count), the starting index (Start)
57      # or the template used to construct the name (Hostname).
58      #
59      # Note: Template and Specs are not mutually exclusive.  You may define both
60      # sections and the aggregate nodes will be created for you.  Take care with
61      # name collisions
62      # ---------------------------------------------------------------------------
63      Template:
64        Count: 2
65        # Start: 5
66        # Hostname: {{.Prefix}}{{.Index}} # default
67      # ---------------------------------------------------------------------------
68      # "Users"
69      # ---------------------------------------------------------------------------
70      # Count: The number of user accounts _in addition_ to Admin
71      # ---------------------------------------------------------------------------
72      Users:
73        Count: 1
74    # ---------------------------------------------------------------------------
75    # Org2: See "Org1" for full specification
76    # ---------------------------------------------------------------------------
77    - Name: Org2
78      Domain: org2.example.com
79      EnableNodeOUs: true
80      Template:
81        Count: 2
82      Users:
83        Count: 1
```

A.3.2 docker-compose-cli.yaml

docker-compose-cli.yaml [36] is included in the first-network folder in the Hyperledger example files. It is used to define the Docker containers that will each peer and order on the network will be hosted. The code is modified from the source on lines 82 and 84 to point to a different script and chaincode folders.

```
1  # Copyright IBM Corp. All Rights Reserved.
2  #
3  # SPDX-License-Identifier: Apache-2.0
4  #
5
6  version: '2'
7
8  volumes:
```

```
    orderer.example.com:
    peer0.org1.example.com:
    peer1.org1.example.com:
    peer0.org2.example.com:
    peer1.org2.example.com:

networks:
  byfn:

services:

  orderer.example.com:
    extends:
      file:   base/docker-compose-base.yaml
      service: orderer.example.com
    container_name: orderer.example.com
    networks:
      - byfn

  peer0.org1.example.com:
    container_name: peer0.org1.example.com
    extends:
      file:  base/docker-compose-base.yaml
      service: peer0.org1.example.com
    networks:
      - byfn

  peer1.org1.example.com:
    container_name: peer1.org1.example.com
    extends:
      file:  base/docker-compose-base.yaml
      service: peer1.org1.example.com
    networks:
      - byfn

  peer0.org2.example.com:
    container_name: peer0.org2.example.com
    extends:
      file:  base/docker-compose-base.yaml
      service: peer0.org2.example.com
    networks:
      - byfn

  peer1.org2.example.com:
    container_name: peer1.org2.example.com
    extends:
      file:  base/docker-compose-base.yaml
      service: peer1.org2.example.com
    networks:
      - byfn

  cli:
```

```
61      container_name: cli
62      image: hyperledger/fabric-tools:$IMAGE_TAG
63      tty: true
64      stdin_open: true
65      environment:
66        - GOPATH=/opt/gopath
67        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
68        #- CORE_LOGGING_LEVEL=DEBUG
69        - CORE_LOGGING_LEVEL=INFO
70        - CORE_PEER_ID=cli
71        - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
72        - CORE_PEER_LOCALMSPID=Org1MSP
73        - CORE_PEER_TLS_ENABLED=true
74        - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric \
          ↪   /peer/crypto/peerOrganizations/org1.example.com/peers \
          ↪   /peer0.org1.example.com/tls/server.crt
75        - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric \
          ↪   /peer/crypto/peerOrganizations/org1.example.com/peers \
          ↪   /peer0.org1.example.com/tls/server.key
76        - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric \
          ↪   /peer/crypto/peerOrganizations/org1.example.com/peers \ /peer0.org1.example.com/tls/ca.crt
77        - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric \
          ↪   /peer/crypto/peerOrganizations/org1.example.com/users \ /Admin@org1.example.com/msp
78      working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
79  #command: /bin/bash
80      volumes:
81          - /var/run/:/host/var/run/
82          - ./../thesis/chaincode/:/opt/gopath/src/github.com/hyperledger/fabric \ /examples/chaincode/go
83          - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
84          - ./../thesis/scripts/:/opt/gopath/src/github.com/hyperledger/fabric \ /peer/scripts/
85          - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric \ /peer/channel-artifacts
86      depends_on:
87        - orderer.example.com
88        - peer0.org1.example.com
89        - peer1.org1.example.com
90        - peer0.org2.example.com
91        - peer1.org2.example.com
92      networks:
93        - byfn
```

A.3.3 docker-compose-couch.yaml

docker-compose-couch.yaml [36] is included in the first-network folder in the Hyperledger example files. It is used to define the settings for a Docker container that will host the CouchDB, NoSQL database. The code is presented here unmodified from the source.

```yaml
# Copyright IBM Corp. All Rights Reserved.
#
# SPDX-License-Identifier: Apache-2.0
#

version: '2'

networks:
  byfn:

services:
  couchdb0:
    container_name: couchdb0
    image: hyperledger/fabric-couchdb
    # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and password
    # for CouchDB.  This will prevent CouchDB from operating in an "Admin Party" mode.
    environment:
      - COUCHDB_USER=
      - COUCHDB_PASSWORD=
    # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
    # for example map it to utilize Fauxton User Interface in dev environments.
    ports:
      - "5984:5984"
    networks:
      - byfn

  peer0.org1.example.com:
    environment:
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
      # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
      # provide the credentials for ledger to connect to CouchDB.  The username and password must
      # match the username and password set for the associated CouchDB.
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
    depends_on:
      - couchdb0

  couchdb1:
    container_name: couchdb1
    image: hyperledger/fabric-couchdb
    # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and password
    # for CouchDB.  This will prevent CouchDB from operating in an "Admin Party" mode.
    environment:
      - COUCHDB_USER=
      - COUCHDB_PASSWORD=
    # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
    # for example map it to utilize Fauxton User Interface in dev environments.
    ports:
      - "6984:5984"
    networks:
```

```
52        - byfn
53
54    peer1.org1.example.com:
55      environment:
56        - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
57        - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
58        # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
59        # provide the credentials for ledger to connect to CouchDB.  The username and password must
60        # match the username and password set for the associated CouchDB.
61        - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
62        - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
63      depends_on:
64        - couchdb1
65
66    couchdb2:
67      container_name: couchdb2
68      image: hyperledger/fabric-couchdb
69      # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and password
70      # for CouchDB.  This will prevent CouchDB from operating in an "Admin Party" mode.
71      environment:
72        - COUCHDB_USER=
73        - COUCHDB_PASSWORD=
74      # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
75      # for example map it to utilize Fauxton User Interface in dev environments.
76      ports:
77        - "7984:5984"
78      networks:
79        - byfn
80
81    peer0.org2.example.com:
82      environment:
83        - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
84        - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984
85        # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
86        # provide the credentials for ledger to connect to CouchDB.  The username and password must
87        # match the username and password set for the associated CouchDB.
88        - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
89        - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
90      depends_on:
91        - couchdb2
92
93    couchdb3:
94      container_name: couchdb3
95      image: hyperledger/fabric-couchdb
96      # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and password
97      # for CouchDB.  This will prevent CouchDB from operating in an "Admin Party" mode.
98      environment:
99        - COUCHDB_USER=
100       - COUCHDB_PASSWORD=
101     # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
102     # for example map it to utilize Fauxton User Interface in dev environments.
103     ports:
```

```
104       - "8984:5984"
105     networks:
106       - byfn
107
108   peer1.org2.example.com:
109     environment:
110       - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
111       - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb3:5984
112       # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
113       # provide the credentials for ledger to connect to CouchDB.  The username and password must
114       # match the username and password set for the associated CouchDB.
115       - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
116       - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
117     depends_on:
118       - couchdb3
```

A.3.4 configtx.yaml

configtx.yaml [36] is included in the first-network folder in the Hyperledger example files. The profiles in configtx.yaml are used configtxgen tool to create configuration artifacts for orderer genesis block, channel configuration, and anchor peers (peers on the network connected to a Hyperledger Fabric channel and able to communicate with other organizations). The code is presented here unmodified from the source with the exception of an Org3 MSP. Line 110 allows for selection of the orderer type with solo and Kafka as currently available options.

```
1  # Copyright IBM Corp. All Rights Reserved.
2  #
3  # SPDX-License-Identifier: Apache-2.0
4  #
5
6  ---
7  ################################################################################
8  #
9  #   Profile
10 #
11 #   - Different configuration profiles may be encoded here to be specified
12 #   as parameters to the configtxgen tool
13 #
14 ################################################################################
15 Profiles:
16
17     TwoOrgsOrdererGenesis:
18         Capabilities:
```

```
            <<: *ChannelCapabilities
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *Org1
                    - *Org2
                    - *Org3

    TwoOrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
                - *Org2
                - *Org3
            Capabilities:
                <<: *ApplicationCapabilities

################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:

    # SampleOrg defines an MSP using the sampleconfig.  It should never be used
    # in production but may be used as a template for other definitions
    - &OrdererOrg
        # DefaultOrg defines the organization which is used in the sampleconfig
        # of the fabric.git development environment
        Name: OrdererOrg

        # ID to load the MSP definition as
        ID: OrdererMSP

        # MSPDir is the filesystem path which contains the MSP configuration
        MSPDir: crypto-config/ordererOrganizations/example.com/msp

    - &Org1
        # DefaultOrg defines the organization which is used in the sampleconfig
        # of the fabric.git development environment
        Name: Org1MSP
```

```
71
72          # ID to load the MSP definition as
73          ID: Org1MSP
74
75          MSPDir: crypto-config/peerOrganizations/org1.example.com/msp
76
77          AnchorPeers:
78              # AnchorPeers defines the location of peers which can be used
79              # for cross org gossip communication.  Note, this value is only
80              # encoded in the genesis block in the Application section context
81              - Host: peer0.org1.example.com
82                Port: 7051
83
84      - &Org2
85          # DefaultOrg defines the organization which is used in the sampleconfig
86          # of the fabric.git development environment
87          Name: Org2MSP
88
89          # ID to load the MSP definition as
90          ID: Org2MSP
91
92          MSPDir: crypto-config/peerOrganizations/org2.example.com/msp
93
94          AnchorPeers:
95              # AnchorPeers defines the location of peers which can be used
96              # for cross org gossip communication.  Note, this value is only
97              # encoded in the genesis block in the Application section context
98              - Host: peer0.org2.example.com
99                Port: 7051
100
101     - &Org3
102         # DefaultOrg defines the organization which is used in the sampleconfig
103         # of the fabric.git development environment
104         Name: Org3MSP
105
106         # ID to load the MSP definition as
107         ID: Org3MSP
108
109         MSPDir: crypto-config/peerOrganizations/org3.example.com/msp
110
111         AnchorPeers:
112             # AnchorPeers defines the location of peers which can be used
113             # for cross org gossip communication.  Note, this value is only
114             # encoded in the genesis block in the Application section context
115             - Host: peer0.org3.example.com
116               Port: 7051
117
118 ################################################################################
119 #
120 #   SECTION: Orderer
121 #
122 #   - This section defines the values to encode into a config transaction or
```

```
#   genesis block for orderer related parameters
#
################################################################################
Orderer: &OrdererDefaults

    # Orderer Type: The orderer implementation to start
    # Available types are "solo" and "kafka"
    OrdererType: solo

    Addresses:
        - orderer.example.com:7050

    # Batch Timeout: The amount of time to wait before creating a batch
    BatchTimeout: 2s

    # Batch Size: Controls the number of messages batched into a block
    BatchSize:

        # Max Message Count: The maximum number of messages to permit in a batch
        MaxMessageCount: 10

        # Absolute Max Bytes: The absolute maximum number of bytes allowed for
        # the serialized messages in a batch.
        AbsoluteMaxBytes: 99 MB

        # Preferred Max Bytes: The preferred maximum number of bytes allowed for
        # the serialized messages in a batch. A message larger than the preferred
        # max bytes will result in a batch larger than preferred max bytes.
        PreferredMaxBytes: 512 KB

    Kafka:
        # Brokers: A list of Kafka brokers to which the orderer connects
        # NOTE: Use IP:port notation
        Brokers:
            - 127.0.0.1:9092

    # Organizations is the list of orgs which are defined as participants on
    # the orderer side of the network
    Organizations:

################################################################################
#
#   SECTION: Application
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for application related parameters
#
################################################################################
Application: &ApplicationDefaults

    # Organizations is the list of orgs which are defined as participants on
    # the application side of the network
```

```
    Organizations:

################################################################################
#
#   SECTION: Capabilities
#
#   - This section defines the capabilities of fabric network. This is a new
#   concept as of v1.1.0 and should not be utilized in mixed networks with
#   v1.0.x peers and orderers.  Capabilities define features which must be
#   present in a fabric binary for that binary to safely participate in the
#   fabric network.  For instance, if a new MSP type is added, newer binaries
#   might recognize and validate the signatures from this type, while older
#   binaries without this support would be unable to validate those
#   transactions.  This could lead to different versions of the fabric binaries
#   having different world states.  Instead, defining a capability for a channel
#   informs those binaries without this capability that they must cease
#   processing transactions until they have been upgraded.  For v1.0.x if any
#   capabilities are defined (including a map with all capabilities turned off)
#   then the v1.0.x peer will deliberately crash.
#
################################################################################
Capabilities:
    # Channel capabilities apply to both the orderers and the peers and must be
    # supported by both.  Set the value of the capability to true to require it.
    Global: &ChannelCapabilities
        # V1.1 for Global is a catchall flag for behavior which has been
        # determined to be desired for all orderers and peers running v1.0.x,
        # but the modification of which would cause incompatibilities.  Users
        # should leave this flag set to true.
        V1_1: true

    # Orderer capabilities apply only to the orderers, and may be safely
    # manipulated without concern for upgrading peers.  Set the value of the
    # capability to true to require it.
    Orderer: &OrdererCapabilities
        # V1.1 for Order is a catchall flag for behavior which has been
        # determined to be desired for all orderers running v1.0.x, but the
        # modification of which  would cause incompatibilities.  Users should
        # leave this flag set to true.
        V1_1: true

    # Application capabilities apply only to the peer network, and may be safely
    # manipulated without concern for upgrading orderers.  Set the value of the
    # capability to true to require it.
    Application: &ApplicationCapabilities
        # V1.1 for Application is a catchall flag for behavior which has been
        # determined to be desired for all peers running v1.0.x, but the
        # modification of which would cause incompatibilities.  Users should
        # leave this flag set to true.
        V1_1: true
```

A.4 Building Hyperledger Fabric

A.4.1 byfn.sh

byfn.sh [36] is a script used to execute the basic commands of up, down, generate, restart, and upgrade the Hyperledger Fabric test network. byfn.sh is presented unmodified with the exception of the addition of Org3 (Lines 427-439). The result of running ./byfn.sh up will consume the .yaml files, generate the certificates and channels and spawn docker containers which run the peers, clients, orderer, CLI, and state databases.

```
1  #!/bin/bash
2  #
3  # Copyright IBM Corp All Rights Reserved
4  #
5  # SPDX-License-Identifier: Apache-2.0
6  #
7
8  # This script will orchestrate a sample end-to-end execution of the Hyperledger
9  # Fabric network.
10 #
11 # The end-to-end verification provisions a sample Fabric network consisting of
12 # two organizations, each maintaining two peers, and a “solo” ordering service.
13 #
14 # This verification makes use of two fundamental tools, which are necessary to
15 # create a functioning transactional network with digital signature validation
16 # and access control:
17 #
18 # * cryptogen - generates the x509 certificates used to identify and
19 #   authenticate the various components in the network.
20 # * configtxgen - generates the requisite configuration artifacts for orderer
21 #   bootstrap and channel creation.
22 #
23 # Each tool consumes a configuration yaml file, within which we specify the topology
24 # of our network (cryptogen) and the location of our certificates for various
25 # configuration operations (configtxgen).  Once the tools have been successfully run,
26 # we are able to launch our network.  More detail on the tools and the structure of
27 # the network will be provided later in this document.  For now, let's get going...
28
29 # prepending $PWD/../bin to PATH to ensure we are picking up the correct binaries
30 # this may be commented out to resolve installed version of tools if desired
31 export PATH=${PWD}/../bin:${PWD}:$PATH
32 export FABRIC_CFG_PATH=${PWD}
33
34 # Print the usage message
35 function printHelp () {
36   echo "Usage: "
```

```
37    echo "  byfn.sh up|down|restart|generate|upgrade [-c <channel name>] [-t <timeout>] [-d <delay>] [-f
       ↪   <docker-compose-file>] [-s <dbtype>] [-i <imagetag>]"
38    echo "  byfn.sh -h|--help (print this message)"
39    echo "    <mode> - one of 'up', 'down', 'restart' or 'generate'"
40    echo "      - 'up' - bring up the network with docker-compose up"
41    echo "      - 'down' - clear the network with docker-compose down"
42    echo "      - 'restart' - restart the network"
43    echo "      - 'generate' - generate required certificates and genesis block"
44    echo "      - 'upgrade'  - upgrade the network from v1.0.x to v1.1"
45    echo "    -c <channel name> - channel name to use (defaults to \"mychannel\")"
46    echo "    -t <timeout> - CLI timeout duration in seconds (defaults to 10)"
47    echo "    -d <delay> - delay duration in seconds (defaults to 3)"
48    echo "    -f <docker-compose-file> - specify which docker-compose file use (defaults to
       ↪   docker-compose-cli.yaml)"
49    echo "    -s <dbtype> - the database backend to use: goleveldb (default) or couchdb"
50    echo "    -l <language> - the chaincode language: golang (default) or node"
51    echo "    -i <imagetag> - the tag to be used to launch the network (defaults to \"latest\")"
52    echo
53    echo "Typically, one would first generate the required certificates and "
54    echo "genesis block, then bring up the network. e.g.:"
55    echo
56    echo "        byfn.sh generate -c mychannel"
57    echo "        byfn.sh up -c mychannel -s couchdb"
58    echo "        byfn.sh up -c mychannel -s couchdb -i 1.1.0-alpha"
59    echo "        byfn.sh up -l node"
60    echo "        byfn.sh down -c mychannel"
61    echo "        byfn.sh upgrade -c mychannel"
62    echo
63    echo "Taking all defaults:"
64    echo "        byfn.sh generate"
65    echo "        byfn.sh up"
66    echo "        byfn.sh down"
67  }
68
69  # Ask user for confirmation to proceed
70  function askProceed () {
71    read -p "Continue? [Y/n] " ans
72    case "$ans" in
73      y|Y|"" )
74        echo "proceeding ..."
75      ;;
76      n|N )
77        echo "exiting..."
78        exit 1
79      ;;
80      * )
81        echo "invalid response"
82        askProceed
83      ;;
84    esac
85  }
86
```

```
# Obtain CONTAINER_IDS and remove them
# TODO Might want to make this optional - could clear other containers
function clearContainers () {
  CONTAINER_IDS=$(docker ps -aq)
  if [ -z "$CONTAINER_IDS" -o "$CONTAINER_IDS" == " " ]; then
    echo "---- No containers available for deletion ----"
  else
    docker rm -f $CONTAINER_IDS
  fi
}

# Delete any images that were generated as a part of this setup
# specifically the following images are often left behind:
# TODO list generated image naming patterns
function removeUnwantedImages() {
  DOCKER_IMAGE_IDS=$(docker images | grep "dev\|none\|test-vp\|peer[0-9]-" | awk '{print $3}')
  if [ -z "$DOCKER_IMAGE_IDS" -o "$DOCKER_IMAGE_IDS" == " " ]; then
    echo "---- No images available for deletion ----"
  else
    docker rmi -f $DOCKER_IMAGE_IDS
  fi
}

# Versions of fabric known not to work with this release of first-network
BLACKLISTED_VERSIONS="^1\.0\. ^1\.1\.0-preview ^1\.1\.0-alpha"

# Do some basic sanity checking to make sure that the appropriate versions of fabric
# binaries/images are available.  In the future, additional checking for the presence
# of go or other items could be added.
function checkPrereqs() {
  # Note, we check configtxlator externally because it does not require a config file, and peer in the
  # docker image because of FAB-8551 that makes configtxlator return 'development version' in docker
  LOCAL_VERSION=$(configtxlator version | sed -ne 's/ Version: //p')
  DOCKER_IMAGE_VERSION=$(docker run --rm hyperledger/fabric-tools:$IMAGETAG peer version | sed -ne 's/
    ↪   Version: //p'|head -1)

  echo "LOCAL_VERSION=$LOCAL_VERSION"
  echo "DOCKER_IMAGE_VERSION=$DOCKER_IMAGE_VERSION"

  if [ "$LOCAL_VERSION" != "$DOCKER_IMAGE_VERSION" ] ; then
     echo "================== WARNING =================="
     echo "  Local fabric binaries and docker images are  "
     echo "  out of  sync. This may cause problems.       "
     echo "============================================="
  fi

  for UNSUPPORTED_VERSION in $BLACKLISTED_VERSIONS ; do
     echo "$LOCAL_VERSION" | grep -q $UNSUPPORTED_VERSION
     if [ $? -eq 0 ] ; then
       echo "ERROR! Local Fabric binary version of $LOCAL_VERSION does not match this newer version of
         ↪   BYFN and is unsupported. Either move to a later version of Fabric or checkout an earlier
         ↪   version of fabric-samples."
```

```
136         exit 1
137       fi
138
139       echo "$DOCKER_IMAGE_VERSION" | grep -q $UNSUPPORTED_VERSION
140       if [ $? -eq 0 ] ; then
141         echo "ERROR! Fabric Docker image version of $DOCKER_IMAGE_VERSION does not match this newer
            ↪   version of BYFN and is unsupported. Either move to a later version of Fabric or checkout
            ↪   an earlier version of fabric-samples."
142         exit 1
143       fi
144     done
145   }
146
147   # Generate the needed certificates, the genesis block and start the network.
148   function networkUp () {
149     checkPrereqs
150     # generate artifacts if they don't exist
151     if [ ! -d "crypto-config" ]; then
152       generateCerts
153       replacePrivateKey
154       generateChannelArtifacts
155     fi
156     if [ "${IF_COUCHDB}" == "couchdb" ]; then
157       IMAGE_TAG=$IMAGETAG docker-compose -f $COMPOSE_FILE -f $COMPOSE_FILE_COUCH up -d 2>&1
158     else
159       IMAGE_TAG=$IMAGETAG docker-compose -f $COMPOSE_FILE up -d 2>&1
160     fi
161     if [ $? -ne 0 ]; then
162       echo "ERROR !!!! Unable to start network"
163       exit 1
164     fi
165     # now run the end to end script
166     #docker exec cli scripts/script.sh $CHANNEL_NAME $CLI_DELAY $LANGUAGE $CLI_TIMEOUT
167     if [ $? -ne 0 ]; then
168       echo "ERROR !!!! Test failed"
169       exit 1
170     fi
171   }
172
173   # Upgrade the network from v1.0.x to v1.1
174   # Stop the orderer and peers, backup the ledger from orderer and peers, cleanup chaincode containers
       ↪   and images
175   # and relaunch the orderer and peers with latest tag
176   function upgradeNetwork () {
177     docker inspect  -f '{{.Config.Volumes}}' orderer.example.com |grep -q
        ↪   '/var/hyperledger/production/orderer'
178     if [ $? -ne 0 ]; then
179       echo "ERROR !!!! This network does not appear to be using volumes for its ledgers, did you start
          ↪   from fabric-samples >= v1.0.6?"
180       exit 1
181     fi
182
```

```
183     LEDGERS_BACKUP=./ledgers-backup
184
185     # create ledger-backup directory
186     mkdir -p $LEDGERS_BACKUP
187
188     export IMAGE_TAG=$IMAGETAG
189     if [ "${IF_COUCHDB}" == "couchdb" ]; then
190         COMPOSE_FILES="-f $COMPOSE_FILE -f $COMPOSE_FILE_COUCH"
191     else
192         COMPOSE_FILES="-f $COMPOSE_FILE"
193     fi
194
195     # removing the cli container
196     docker-compose $COMPOSE_FILES stop cli
197     docker-compose $COMPOSE_FILES up -p 8081:8081 -d --no-deps cli
198
199     echo "Upgrading orderer"
200     docker-compose $COMPOSE_FILES stop orderer.example.com
201     docker cp -a orderer.example.com:/var/hyperledger/production/orderer
        ↪   $LEDGERS_BACKUP/orderer.example.com
202     docker-compose $COMPOSE_FILES up -d --no-deps orderer.example.com
203
204     for PEER in peer0.org1.example.com peer1.org1.example.com peer0.org2.example.com
        ↪   peer1.org2.example.com; do
205       echo "Upgrading peer $PEER"
206
207       # Stop the peer and backup its ledger
208       docker-compose $COMPOSE_FILES stop $PEER
209       docker cp -a $PEER:/var/hyperledger/production $LEDGERS_BACKUP/$PEER/
210
211       # Remove any old containers and images for this peer
212       CC_CONTAINERS=$(docker ps | grep dev-$PEER | awk '{print $1}')
213       if [ -n "$CC_CONTAINERS" ] ; then
214           docker rm -f $CC_CONTAINERS
215       fi
216       CC_IMAGES=$(docker images | grep dev-$PEER | awk '{print $1}')
217       if [ -n "$CC_IMAGES" ] ; then
218           docker rmi -f $CC_IMAGES
219       fi
220
221       # Start the peer again
222       docker-compose $COMPOSE_FILES up -d --no-deps $PEER
223     done
224
225     docker exec cli scripts/upgrade_to_v11.sh $CHANNEL_NAME $CLI_DELAY $LANGUAGE $CLI_TIMEOUT
226     if [ $? -ne 0 ]; then
227       echo "ERROR !!!! Test failed"
228       exit 1
229     fi
230   }
231
232
```

```
233 # Tear down running network
234 function networkDown () {
235   docker-compose -f $COMPOSE_FILE -f $COMPOSE_FILE_COUCH down --volumes
236   docker-compose -f $COMPOSE_FILE down --volumes
237   # Don't remove the generated artifacts -- note, the ledgers are always removed
238   if [ "$MODE" != "restart" ]; then
239     # Bring down the network, deleting the volumes
240     #Delete any ledger backups
241     docker run -v $PWD:/tmp/first-network --rm hyperledger/fabric-tools:$IMAGETAG rm -Rf
       ↪   /tmp/first-network/ledgers-backup
242     #Cleanup the chaincode containers
243     clearContainers
244     #Cleanup images
245     removeUnwantedImages
246     # remove orderer block and other channel configuration transactions and certs
247     rm -rf channel-artifacts/*.block channel-artifacts/*.tx crypto-config
       ↪   ./org3-artifacts/crypto-config/ channel-artifacts/org3.json
248     # remove the docker-compose yaml file that was customized to the example
249     rm -f docker-compose-e2e.yaml
250   fi
251 }
252
253 # Using docker-compose-e2e-template.yaml, replace constants with private key file names
254 # generated by the cryptogen tool and output a docker-compose.yaml specific to this
255 # configuration
256 function replacePrivateKey () {
257   # sed on MacOSX does not support -i flag with a null extension. We will use
258   # 't' for our back-up's extension and depete it at the end of the function
259   ARCH=`uname -s | grep Darwin`
260   if [ "$ARCH" == "Darwin" ]; then
261     OPTS="-it"
262   else
263     OPTS="-i"
264   fi
265
266   # Copy the template to the file that will be modified to add the private key
267   cp docker-compose-e2e-template.yaml docker-compose-e2e.yaml
268
269   # The next steps will replace the template's contents with the
270   # actual values of the private key file names for the two CAs.
271   CURRENT_DIR=$PWD
272   cd crypto-config/peerOrganizations/org1.example.com/ca/
273   PRIV_KEY=$(ls *_sk)
274   cd "$CURRENT_DIR"
275   sed $OPTS "s/CA1_PRIVATE_KEY/${PRIV_KEY}/g" docker-compose-e2e.yaml
276   cd crypto-config/peerOrganizations/org2.example.com/ca/
277   PRIV_KEY=$(ls *_sk)
278   cd "$CURRENT_DIR"
279   sed $OPTS "s/CA2_PRIVATE_KEY/${PRIV_KEY}/g" docker-compose-e2e.yaml
280   # If MacOSX, remove the temporary backup of the docker-compose file
281   if [ "$ARCH" == "Darwin" ]; then
282     rm docker-compose-e2e.yamlt
```

```
283   fi
284 }
285
286 # We will use the cryptogen tool to generate the cryptographic material (x509 certs)
287 # for our various network entities.  The certificates are based on a standard PKI
288 # implementation where validation is achieved by reaching a common trust anchor.
289 #
290 # Cryptogen consumes a file - ``crypto-config.yaml`` - that contains the network
291 # topology and allows us to generate a library of certificates for both the
292 # Organizations and the components that belong to those Organizations.  Each
293 # Organization is provisioned a unique root certificate (``ca-cert``), that binds
294 # specific components (peers and orderers) to that Org.  Transactions and communications
295 # within Fabric are signed by an entity's private key (``keystore``), and then verified
296 # by means of a public key (``signcerts``).  You will notice a "count" variable within
297 # this file.  We use this to specify the number of peers per Organization; in our
298 # case it's two peers per Org.  The rest of this template is extremely
299 # self-explanatory.
300 #
301 # After we run the tool, the certs will be parked in a folder titled ``crypto-config``.
302
303 # Generates Org certs using cryptogen tool
304 function generateCerts (){
305   which cryptogen
306   if [ "$?" -ne 0 ]; then
307     echo "cryptogen tool not found. exiting"
308     exit 1
309   fi
310   echo
311   echo "##########################################################"
312   echo "##### Generate certificates using cryptogen tool #########"
313   echo "##########################################################"
314
315   if [ -d "crypto-config" ]; then
316     rm -Rf crypto-config
317   fi
318   set -x
319   cryptogen generate --config=./crypto-config.yaml
320   res=$?
321   set +x
322   if [ $res -ne 0 ]; then
323     echo "Failed to generate certificates..."
324     exit 1
325   fi
326   echo
327 }
328
329 # The `configtxgen tool is used to create four artifacts: orderer **bootstrap
330 # block**, fabric **channel configuration transaction**, and two **anchor
331 # peer transactions** - one for each Peer Org.
332 #
333 # The orderer block is the genesis block for the ordering service, and the
334 # channel transaction file is broadcast to the orderer at channel creation
```

```
335 # time.  The anchor peer transactions, as the name might suggest, specify each
336 # Org's anchor peer on this channel.
337 #
338 # Configtxgen consumes a file - ``configtx.yaml`` - that contains the definitions
339 # for the sample network. There are three members - one Orderer Org (``OrdererOrg``)
340 # and two Peer Orgs (``Org1`` & ``Org2``) each managing and maintaining two peer nodes.
341 # This file also specifies a consortium - ``SampleConsortium`` - consisting of our
342 # two Peer Orgs.  Pay specific attention to the "Profiles" section at the top of
343 # this file.  You will notice that we have two unique headers. One for the orderer genesis
344 # block - ``TwoOrgsOrdererGenesis`` - and one for our channel - ``TwoOrgsChannel``.
345 # These headers are important, as we will pass them in as arguments when we create
346 # our artifacts.  This file also contains two additional specifications that are worth
347 # noting.  Firstly, we specify the anchor peers for each Peer Org
348 # (``peer0.org1.example.com`` & ``peer0.org2.example.com``).  Secondly, we point to
349 # the location of the MSP directory for each member, in turn allowing us to store the
350 # root certificates for each Org in the orderer genesis block.  This is a critical
351 # concept. Now any network entity communicating with the ordering service can have
352 # its digital signature verified.
353 #
354 # This function will generate the crypto material and our four configuration
355 # artifacts, and subsequently output these files into the ``channel-artifacts``
356 # folder.
357 #
358 # If you receive the following warning, it can be safely ignored:
359 #
360 # [bccsp] GetDefault -> WARN 001 Before using BCCSP, please call InitFactories(). Falling back to
     ↪  bootBCCSP.
361 #
362 # You can ignore the logs regarding intermediate certs, we are not using them in
363 # this crypto implementation.
364
365 # Generate orderer genesis block, channel configuration transaction and
366 # anchor peer update transactions
367 function generateChannelArtifacts() {
368   which configtxgen
369   if [ "$?" -ne 0 ]; then
370     echo "configtxgen tool not found. exiting"
371     exit 1
372   fi
373
374   echo "#########################################################"
375   echo "#########  Generating Orderer Genesis block ##############"
376   echo "#########################################################"
377   # Note: For some unknown reason (at least for now) the block file can't be
378   # named orderer.genesis.block or the orderer will fail to launch!
379   set -x
380   configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block
381   res=$?
382   set +x
383   if [ $res -ne 0 ]; then
384     echo "Failed to generate orderer genesis block..."
385     exit 1
```

```
386     fi
387     echo
388     echo "#################################################################"
389     echo "### Generating channel configuration transaction 'channel.tx' ###"
390     echo "#################################################################"
391     set -x
392     configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID
         ↪   $CHANNEL_NAME
393     res=$?
394     set +x
395     if [ $res -ne 0 ]; then
396       echo "Failed to generate channel configuration transaction..."
397       exit 1
398     fi
399
400     echo
401     echo "#################################################################"
402     echo "#######    Generating anchor peer update for Org1MSP   ##########"
403     echo "#################################################################"
404     set -x
405     configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx
         ↪   -channelID $CHANNEL_NAME -asOrg Org1MSP
406     res=$?
407     set +x
408     if [ $res -ne 0 ]; then
409       echo "Failed to generate anchor peer update for Org1MSP..."
410       exit 1
411     fi
412
413     echo
414     echo "#################################################################"
415     echo "#######    Generating anchor peer update for Org2MSP   ##########"
416     echo "#################################################################"
417     set -x
418     configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate \
419     ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org2MSP
420     res=$?
421     set +x
422     if [ $res -ne 0 ]; then
423       echo "Failed to generate anchor peer update for Org2MSP..."
424       exit 1
425     fi
426
427     echo
428     echo "#################################################################"
429     echo "#######    Generating anchor peer update for Org3MSP   ##########"
430     echo "#################################################################"
431     set -x
432     configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate \
433     ./channel-artifacts/Org3MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org3MSP
434     res=$?
435     set +x
```

```
436     if [ $res -ne 0 ]; then
437       echo "Failed to generate anchor peer update for Org3MSP..."
438       exit 1
439     fi
440     echo
441   }
442
443   # Obtain the OS and Architecture string that will be used to select the correct
444   # native binaries for your platform
445   OS_ARCH=$(echo "$(uname -s|tr '[:upper:]' '[:lower:]'|sed 's/mingw64_nt.*/windows/')-$(uname -m | sed
       ↪   's/x86_64/amd64/g')" | awk '{print tolower($0)}')
446   # timeout duration - the duration the CLI should wait for a response from
447   # another container before giving up
448   CLI_TIMEOUT=10
449   # default for delay between commands
450   CLI_DELAY=3
451   # channel name defaults to "mychannel"
452   CHANNEL_NAME="mychannel"
453   # use this as the default docker-compose yaml definition
454   COMPOSE_FILE=docker-compose-cli.yaml
455   #
456   COMPOSE_FILE_COUCH=docker-compose-couch.yaml
457   # use golang as the default language for chaincode
458   LANGUAGE=golang
459   # default image tag
460   IMAGETAG="latest"
461   # Parse commandline args
462   if [ "$1" = "-m" ];then        # supports old usage, muscle memory is powerful!
463       shift
464   fi
465   MODE=$1;shift
466   # Determine whether starting, stopping, restarting or generating for announce
467   if [ "$MODE" == "up" ]; then
468     EXPMODE="Starting"
469   elif [ "$MODE" == "down" ]; then
470     EXPMODE="Stopping"
471   elif [ "$MODE" == "restart" ]; then
472     EXPMODE="Restarting"
473   elif [ "$MODE" == "generate" ]; then
474     EXPMODE="Generating certs and genesis block for"
475   elif [ "$MODE" == "upgrade" ]; then
476     EXPMODE="Upgrading the network"
477   else
478     printHelp
479     exit 1
480   fi
481
482   while getopts "h?m:c:t:d:f:s:l:i:" opt; do
483     case "$opt" in
484       h|\?)
485         printHelp
486         exit 0
```

```
    ;;
    c)  CHANNEL_NAME=$OPTARG
    ;;
    t)  CLI_TIMEOUT=$OPTARG
    ;;
    d)  CLI_DELAY=$OPTARG
    ;;
    f)  COMPOSE_FILE=$OPTARG
    ;;
    s)  IF_COUCHDB=$OPTARG
    ;;
    l)  LANGUAGE=$OPTARG
    ;;
    i)  IMAGETAG=`uname -m`"-"$OPTARG
    ;;
  esac
done

# Announce what was requested

  if [ "${IF_COUCHDB}" == "couchdb" ]; then
        echo
        echo "${EXPMODE} with channel '${CHANNEL_NAME}' and CLI timeout of '${CLI_TIMEOUT}' seconds and
         ↪  CLI delay of '${CLI_DELAY}' seconds and using database '${IF_COUCHDB}'"
  else
        echo "${EXPMODE} with channel '${CHANNEL_NAME}' and CLI timeout of '${CLI_TIMEOUT}' seconds and
         ↪  CLI delay of '${CLI_DELAY}' seconds"
  fi
# ask for confirmation to proceed
askProceed

#Create the network using docker compose
if [ "${MODE}" == "up" ]; then
  networkUp
elif [ "${MODE}" == "down" ]; then ## Clear the network
  networkDown
elif [ "${MODE}" == "generate" ]; then ## Generate Artifacts
  generateCerts
  replacePrivateKey
  generateChannelArtifacts
elif [ "${MODE}" == "restart" ]; then ## Restart the network
  networkDown
  networkUp
elif [ "${MODE}" == "upgrade" ]; then ## Upgrade the network from v1.0.x to v1.1
  upgradeNetwork
else
  printHelp
  exit 1
fi
```

A.5 Hyperledger Scripts

The Hyperledger Fabric scripts are run on the CLI docker container that is created using byfn.sh.

A.5.1 setclienv.sh

setclienv.sh is used to set environmental variables used in the other scripts. The code is adapted from [39].

```
export CHANNEL_NAME=sdnnetwork
export CC_NAME=thesischaincode
```

A.5.2 channel-setup.sh

channel-setup.sh is used to create the channel named in setclienv.sh and join the peers to the channel. The code is adapted from [39].

```
ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ \
    ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/ \
    tlsca.example.com-cert.pem

# Channel creation
echo "========== Creating channel: "$CHANNEL_NAME" =========="
peer channel create -o orderer.example.com:7050 -c $CHANNEL_NAME -f ../channel-artifacts/channel.tx
    --tls $CORE_PEER_TLS_ENABLED --cafile
    /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations \
    /example.com/orderers/orderer.example.com/msp/tlscacerts/ \ tlsca.example.com-cert.pem

# peer0.org1 channel join
echo "========== Joining peer0.org1.example.com to channel mychannel =========="
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    /crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=peer0.org1.example.com:7051
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    /crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
peer channel join -b ${CHANNEL_NAME}.block
peer channel update -o orderer.example.com:7050 -c $CHANNEL_NAME -f
    ../channel-artifacts/${CORE_PEER_LOCALMSPID}anchors.tx --tls $CORE_PEER_TLS_ENABLED --cafile
    $ORDERER_CA

```

```
16 # peer1.org1 channel join
17 echo "========== Joining peer1.org1.example.com to channel mychannel =========="
18 export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
19 export CORE_PEER_ADDRESS=peer1.org1.example.com:7051
20 export CORE_PEER_LOCALMSPID="Org1MSP"
21 export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt
22 peer channel join -b ${CHANNEL_NAME}.block
23
24 # peer0.org2 channel join
25 echo "========== Joining peer0.org2.example.com to channel mychannel =========="
26 export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
27 export CORE_PEER_ADDRESS=peer0.org2.example.com:7051
28 export CORE_PEER_LOCALMSPID="Org2MSP"
29 export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt
30 peer channel join -b ${CHANNEL_NAME}.block
31 peer channel update -o orderer.example.com:7050 -c $CHANNEL_NAME -f
    ↪   ../channel-artifacts/${CORE_PEER_LOCALMSPID}anchors.tx --tls $CORE_PEER_TLS_ENABLED --cafile
    ↪   $ORDERER_CA
32
33 # peer1.org2 channel join
34 echo "========== Joining peer1.org2.example.com to channel mychannel =========="
35 export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
36 export CORE_PEER_ADDRESS=peer1.org2.example.com:7051
37 export CORE_PEER_LOCALMSPID="Org2MSP"
38 export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt
39 peer channel join -b ${CHANNEL_NAME}.block
40
41 # peer0.org3 channel join
42 echo "========== Joining peer0.org3.example.com to channel mychannel =========="
43 export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp
44 export CORE_PEER_ADDRESS=peer0.org3.example.com:7051
45 export CORE_PEER_LOCALMSPID="Org3MSP"
46 export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org3.example.com/peers/peer1.org3.example.com/tls/ca.crt
47 peer channel join -b ${CHANNEL_NAME}.block
48 peer channel update -o orderer.example.com:7050 -c $CHANNEL_NAME -f
    ↪   ../channel-artifacts/${CORE_PEER_LOCALMSPID}anchors.tx --tls $CORE_PEER_TLS_ENABLED --cafile
    ↪   $ORDERER_CA
49
50 # peer1.org3 channel join
51 echo "========== Joining peer1.org3.example.com to channel mychannel =========="
52 export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
    ↪   /crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp
53 export CORE_PEER_ADDRESS=peer1.org3.example.com:7051
54 export CORE_PEER_LOCALMSPID="Org3MSP"
```

```
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org3.example.com/peers/peer1.org3.example.com/tls/ca.crt
peer channel join -b ${CHANNEL_NAME}.block
```

A.5.3 install-chaincode.sh

install-chaincode.sh is used to install a copy of the chaincode (contract code) on all peers. The code is adapted from [39].

```
echo "========== Installing chaincode on peer0.org1 =========="
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=peer0.org1.example.com:7051
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer\
  ↪   /crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
peer chaincode install -n $CC_NAME -v $1 -p github.com/hyperledger/fabric/examples/chaincode/go

echo "========== Installing chaincode on peer1.org1 =========="
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=peer1.org1.example.com:7051
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt
peer chaincode install -n $CC_NAME -v $1 -p github.com/hyperledger/fabric/examples/chaincode/go

echo "========== Installing chaincode on peer0.org2 =========="
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=peer0.org2.example.com:7051
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
peer chaincode install -n $CC_NAME -v $1 -p github.com/hyperledger/fabric/examples/chaincode/go

echo "========== Installing chaincode on peer1.org2 =========="
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=peer1.org2.example.com:7051
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt
peer chaincode install -n $CC_NAME -v $1 -p github.com/hyperledger/fabric/examples/chaincode/go

echo "========== Installing chaincode on peer0.org3 =========="
```

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp
export CORE_PEER_ADDRESS=peer0.org3.example.com:7051
export CORE_PEER_LOCALMSPID="Org3MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt
peer chaincode install -n $CC_NAME -v $1 -p github.com/hyperledger/fabric/examples/chaincode/go

echo "========== Installing chaincode on peer1.org3 =========="
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp
export CORE_PEER_ADDRESS=peer1.org3.example.com:7051
export CORE_PEER_LOCALMSPID="Org3MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer \
  ↪   /crypto/peerOrganizations/org3.example.com/peers/peer1.org3.example.com/tls/ca.crt
peer chaincode install -n $CC_NAME -v $1 -p github.com/hyperledger/fabric/examples/chaincode/go
```

A.5.4 instantiate-chaincode.sh

install-chaincode.sh is used to install a copy of the chaincode (contract code) on all peers. The code is adapted from [39].

```
echo "========== Instantiating chaincode v$1 =========="
peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile
  ↪   /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ \
  ↪   example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C
  ↪   $CHANNEL_NAME -n $CC_NAME -v $1 -c '{"Args":["init"]}' -P "OR
  ↪   ('Org0MSP.peer','Org1MSP.peer','Org2MSP.peer')"
```

A.5.5 upgrade-chaincode.sh

upgrade-chaincode.sh is used to upload a new version of contract code once the install command has already been used. The upgrade command updates a copy of the chaincode (contract code) on all peers. The code is adapted from [39].

```
echo "========== Upgrade chaincode to version $1 =========="
peer chaincode upgrade -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
  ↪   /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ \
  ↪   example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C
  ↪   $CHANNEL_NAME -n $CC_NAME -c '{"Args": []}' -v $1 -P "OR ('Org1MSP.member','Org2MSP.member')"
```

A.6 Contract Code

Contract code for SDN/Hyperledger

```
/*
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements.  See the NOTICE file
distributed with this work for additional information
regarding copyright ownership.  The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License.  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.  See the License for the
specific language governing permissions and limitations
under the License.
*/

package main

import (
        "bytes"
        "encoding/json"
        "fmt"
        "strconv"
        "strings"
        "time"
        //"os"

        "github.com/hyperledger/fabric/core/chaincode/shim"
        pb "github.com/hyperledger/fabric/protos/peer"
)

type SimpleChaincode struct {
}

type tenant struct {
        ObjectType string   `json:"docType"`
        Name       string   `json:"name"`
        StartTime  int64    `json:"starttime"`
        Duration   int64    `json:"duration"`
        Bandwidth  float64  `json:"bandwidth"` //Mbps
        Hosts      int      `json:"hosts"`
        SAddress   []string `json:"saddress"`
        DAddress []string `json:"daddress"`  //tagged for removal
        TransNum int      `json:"transnum"`
```

```
48  }
49
50  type network struct {
51          ObjectType  string   `json:"docType"`     //docType distinguishes types of objects in state
             ↪   database
52          Name        string   `json:"name"`        //unique identifier for entry
53          Address     string   `json:"address"`     //network address of sdn network
54          Hosts       int      `json:"hosts"`       //Total hosts
55          Availhosts  string   `json:"availhosts"`  //Currently avaliable hosts
56          Bandwidth   float64  `json:"bandwidth"`   //Avaliable bandwith in Mbps
57          Balance     float64  `json:"balance"`     //Currency ballance of sdn network
58          Pricepermb  float64  `json:"pricepermb"`  //price per megabyte per hour
59          Tenantslice []tenant `json:"tenantslice"` //List of networks with contracts in execution
60          ASnum       int      `json:"asnum"`       //AS number where the network resides
61          ASneighbor  []int    `json:"asneighbor"`  //Neighbor AS numbers
62          TransCount  int      `json:"transcount"`  //number of executed tranactions
63
64  }
65
66  // ============================================================================================
67  // Main
68  // ============================================================================================
69  func main() {
70          err := shim.Start(new(SimpleChaincode))
71          if err != nil {
72                  fmt.Printf("Error starting Simple chaincode: %s", err)
73          }
74  }
75
76  // Init initializes chaincode
77  // ===========================
78  func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
79          return shim.Success(nil)
80  }
81
82  // Invoke - Our entry point for Invocations
83  // ========================================
84  func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
85          function, args := stub.GetFunctionAndParameters()
86          fmt.Println("invoke is running " + function)
87
88          // Handles functions called with Invoike
89          if function == "initNetwork" { //create a new network
90                  return t.initNetwork(stub, args)
91          } else if function == "delete" { //delete a network
92                  return t.delete(stub, args)
93          } else if function == "readNetwork" { //read a network
94                  return t.readNetwork(stub, args)
95          } else if function == "queryAvailhosts" { //find networks for owner X using rich query
96                  return t.queryAvailhosts(stub, args)
97          } else if function == "queryNetwork" { //find networks based on an ad hoc rich query
98                  return t.queryNetwork(stub, args)
```

```
99          } else if function == "queryBandwidth" { //get all networks that have a specified number of
             ↪  bandwidth
100                 return t.queryBandwidth(stub, args)
101         } else if function == "queryPricePerMB" { //get all networks that have a specified price per
             ↪  Megabyte
102                 return t.queryPricePerMB(stub, args)
103         } else if function == "queryHosts" { //get all networks that have a specified number of hosts
104                 return t.queryHosts(stub, args)
105         } else if function == "transactionNetwork" { //get all networks that have a specified number of
             ↪  hosts
106                 return t.transactionNetwork(stub, args)
107         } else if function == "getHistoryForNetwork" { //get history of values for a network
108                 return t.getHistoryForNetwork(stub, args)
109
110         } else if function == "pruneTenants" { //get networks based on range query
111                 return t.pruneTenants(stub, args)
112         }
113
114         fmt.Println("invoke did not find func: " + function) //error
115         return shim.Error("Received unknown function invocation")
116 }
117
118 // ============================================================
119 // transactionNetwork - Checks that all requirements for a tranactin are met
120 // then updates all network values, adds the requesting network as tennant on
121 // the supporting network.  Calculates cost and exchanges currency.
122 // ============================================================
123 func (t *SimpleChaincode) transactionNetwork(stub shim.ChaincodeStubInterface, args []string)
     ↪  pb.Response {
124         var err error
125
126         //   0      1      2        3        4        5         5            6
127         // "net1","net2","dur","Bandwidth","hosts" "saddr" "smac (list)" "daddr (list)"
128         // requesting, accepting
129         if len(args) != 7 {
130                 return shim.Error("Incorrect number of arguments. Expecting 7")
131         }
132
133         // ==== Input sanitation ====
134         fmt.Println("- start init network")
135         if len(args[0]) <= 0 {
136                 return shim.Error("1st argument must be a non-empty string")
137         }
138         if len(args[1]) <= 0 {
139                 return shim.Error("2nd argument must be a non-empty string")
140         }
141         if len(args[2]) <= 0 {
142                 return shim.Error("3rd argument must be a non-empty string")
143         }
144         if len(args[3]) <= 0 {
145                 return shim.Error("4th argument must be a non-empty string")
146         }
```

```
147         if len(args[3]) <= 0 {
148                 return shim.Error("5th argument must be a non-empty string")
149         }
150         if len(args[3]) <= 0 {
151                 return shim.Error("6th argument must be a non-empty string")
152         }
153         if len(args[3]) <= 0 {
154                 return shim.Error("7th argument must be a non-empty string")
155         }
156
157         network1Name := args[0]
158         network2Name := args[1]
159
160         duration, err := strconv.ParseInt(args[2], 10, 64)
161         if err != nil {
162                 return shim.Error("3th argument must be a numeric string")
163         }
164         if duration <= 0 {
165                 return shim.Error("3rd arguement err. duration must be greater than zero")
166         }
167
168         bandwidth, err := strconv.ParseFloat(args[3], 64)
169         if err != nil {
170                 return shim.Error("4th argument must be a numeric string")
171         }
172
173         hosts, err := strconv.Atoi(args[4])
174         if err != nil {
175                 return shim.Error("5rd argument must be a numeric string")
176         }
177
178         TrimArg5 := args[5]
179         TrimArg5 = strings.Trim(TrimArg5, "[")
180         TrimArg5 = strings.Trim(TrimArg5, "]")
181         SAddress := strings.Split(TrimArg5, ",")
182         TrimArg6 := args[6]
183         TrimArg6 = strings.Trim(TrimArg6, "[")
184         TrimArg6 = strings.Trim(TrimArg6, "]")
185         DAddress := strings.Split(TrimArg6, ",")
186
187         //DAddressAsbytes := strings.Split(args[6], ",")
188         var jsonResp string
189         var network1JSON network
190         var network2JSON network
191
192         // ==== Unmarshal data from both networks ====
193
194         valAsbytes, err := stub.GetState(network1Name) //get the network from chaincode state
195         if err != nil {
196                 jsonResp = "{\"Error\":\"Failed to get state for " + network1Name + "\"}"
197                 return shim.Error(jsonResp)
198         } else if valAsbytes == nil {
```

```
199                 jsonResp = "{\"Error\":\"Network does not exist: " + network1Name + "\"}"
200                 return shim.Error(jsonResp)
201         }
202
203         err = json.Unmarshal([]byte(valAsbytes), &network1JSON)
204         if err != nil {
205                 jsonResp = "{\"Error\":\"Failed to decode JSON of: " + network1Name + "\"}"
206                 return shim.Error(jsonResp)
207         }
208
209         valAsbytes2, err := stub.GetState(network2Name) //get the network from chaincode state
210         if err != nil {
211                 jsonResp = "{\"Error\":\"Failed to get state for " + network2Name + "\"}"
212                 return shim.Error(jsonResp)
213         } else if valAsbytes == nil {
214                 jsonResp = "{\"Error\":\"Network does not exist: " + network2Name + "\"}"
215                 return shim.Error(jsonResp)
216         }
217         err = json.Unmarshal([]byte(valAsbytes2), &network2JSON)
218         if err != nil {
219                 jsonResp = "{\"Error\":\"Failed to decode JSON of: " + network2Name + "\"}"
220                 return shim.Error(jsonResp)
221         }
222
223         if network2JSON.Bandwidth < bandwidth {
224                 return shim.Error("bandwidth requested exceeds capacity of " + network2JSON.Name)
225         }
226
227         if network2JSON.Hosts < hosts {
228                 return shim.Error("hosts requested exceeds capacity of " + network2JSON.Name)
229         }
230         durationfloat := float64(duration)
231         cost := (network2JSON.Pricepermb * (durationfloat / 3600)) * bandwidth
232         network2JSON.Bandwidth = network2JSON.Bandwidth - bandwidth
233         network2JSON.Hosts = network2JSON.Hosts - hosts
234         network1JSON.Balance = network1JSON.Balance - cost
235         network2JSON.Balance = network2JSON.Balance + cost
236
237         objectType := "tenant"
238         uniqueid := network2JSON.TransCount + 1
239         network2JSON.TransCount += 1
240         newtenant := tenant{objectType, network1JSON.Name, time.Now().Unix(), duration, bandwidth,
             ↪   hosts, SAddress, DAddress, uniqueid}
241
242         tenantslice := []tenant{}
243
244         for _, item := range network2JSON.Tenantslice {
245                 tenantslice = append(tenantslice, item)
246         }
247         tenantslice = append(tenantslice, newtenant)
248
249         network2JSON.Tenantslice = tenantslice
```

```
250
251         network1JSONasBytes, _ := json.Marshal(network1JSON)
252         err = stub.PutState(network1Name, network1JSONasBytes) //rewrite the network
253         if err != nil {
254                 return shim.Error(err.Error())
255         }
256         network2JSONasBytes, _ := json.Marshal(network2JSON)
257         err = stub.PutState(network2Name, network2JSONasBytes) //rewrite the network
258         if err != nil {
259                 return shim.Error(err.Error())
260         }
261
262         Results := []byte(network1JSON.Name + network2JSON.Name +
          ↪   strconv.FormatFloat(network1JSON.Bandwidth, 'f', -1, 64))
263         return shim.Success(Results)
264         fmt.Println("- end transaction (success)")
265         return shim.Success(nil)
266 }
267
268 // ============================================================
269 // pruneTenants - Checks tennants of the named network to determine if Current
270 // time is > tennant start time + duratin. If so, removes the tennant from the
271 // list and restores bandwidth and host values.
272 // ============================================================
273 func (t *SimpleChaincode) pruneTenants(stub shim.ChaincodeStubInterface, args []string) pb.Response {
274         if len(args) != 1 {
275                 return shim.Error("Incorrect number of arguments. Expecting 1")
276         }
277
278         networkName := args[0]
279
280         // ==== Unmarshal data from both networks ====
281         var jsonResp string
282         var networkJSON network
283
284         valAsbytes, err := stub.GetState(networkName) //get the network from chaincode state
285         if err != nil {
286                 jsonResp = "{\"Error\":\"Failed to get state for " + networkName + "\"}"
287                 return shim.Error(jsonResp)
288         } else if valAsbytes == nil {
289                 jsonResp = "{\"Error\":\"Network does not exist: " + networkName + "\"}"
290                 return shim.Error(jsonResp)
291         }
292
293         err = json.Unmarshal([]byte(valAsbytes), &networkJSON)
294         if err != nil {
295                 jsonResp = "{\"Error\":\"Failed to decode JSON of: " + networkName + "\"}"
296                 return shim.Error(jsonResp)
297         }
298
299         tenantslice := []tenant{}
300         for _, item := range networkJSON.Tenantslice {
```

```
301                 if time.Now().Unix() < item.StartTime+item.Duration {
302                         tenantslice = append(tenantslice, item)
303                 } else {
304                         networkJSON.Bandwidth += item.Bandwidth
305                         networkJSON.Hosts += item.Hosts
306                 }
307         }
308
309         networkJSON.Tenantslice = tenantslice
310
311         networkJSONasBytes, _ := json.Marshal(networkJSON)
312         err = stub.PutState(networkName, networkJSONasBytes)
313         if err != nil {
314                 return shim.Error(err.Error())
315         }
316
317         fmt.Println("- end prune tenants (success)")
318         return shim.Success(nil)
319 }
320
321 // ============================================================
322 // initNetwork - create a new network, store into chaincode state
323 // ============================================================
324 func (t *SimpleChaincode) initNetwork(stub shim.ChaincodeStubInterface, args []string) pb.Response {
325         var err error
326
327         //   0              1        2       3      4          5        6     7        8
328         // "network1", "10.0.0.1", "35",  "true", "3000.0", "100000", "5.2"  "23"    "[22,4,12]"
329         if len(args) != 9 {
330                 return shim.Error("Incorrect number of arguments. Expecting 7")
331         }
332
333         // ==== Input sanitation ====
334         fmt.Println("- start init network")
335         if len(args[0]) <= 0 {
336                 return shim.Error("1st argument must be a non-empty string")
337         }
338         if len(args[1]) <= 0 {
339                 return shim.Error("2nd argument must be a non-empty string")
340         }
341         if len(args[2]) <= 0 {
342                 return shim.Error("3rd argument must be a non-empty string")
343         }
344         if len(args[3]) <= 0 {
345                 return shim.Error("4th argument must be a non-empty string")
346         }
347         if len(args[4]) <= 0 {
348                 return shim.Error("5th argument must be a non-empty string")
349         }
350         if len(args[5]) <= 0 {
351                 return shim.Error("6th argument must be a non-empty string")
352         }
```

```
353         if len(args[6]) <= 0 {
354                 return shim.Error("7th argument must be a non-empty string")
355         }
356         if len(args[7]) <= 0 {
357                 return shim.Error("8th argument must be a non-empty string")
358         }
359         if len(args[8]) <= 0 {
360                 return shim.Error("9th argument must be a non-empty string")
361         }
362         networkName := args[0]
363         address := strings.ToLower(args[1])
364         hosts, err := strconv.Atoi(args[2])
365         if err != nil {
366                 return shim.Error("3rd argument must be a numeric string")
367         }
368         availhosts := strings.ToLower(args[3])
369         if availhosts != "true" {
370                 if availhosts != "false" {
371                         return shim.Error("4th argument must be string `true` or `false`")
372                 }
373         }
374         bandwidth, err := strconv.ParseFloat(args[4], 64)
375         if err != nil {
376                 return shim.Error("5th argument must be a numeric string")
377         }
378         balance, err := strconv.ParseFloat(args[5], 64)
379         if err != nil {
380                 return shim.Error("6th argument must be a numeric string")
381         }
382         pricepermb, err := strconv.ParseFloat(args[6], 64)
383         if err != nil {
384                 return shim.Error("7th argument must be a numeric string")
385         }
386         asnum, err := strconv.Atoi(args[7])
387         if err != nil {
388                 return shim.Error("8rd argument must be a numeric string")
389         }
390         var asneighbor []int
391         err = json.Unmarshal([]byte(args[8]), &asneighbor)
392 
393         if err != nil {
394                 return shim.Error(err.Error())
395         }
396 
397         var tennantslice []tenant
398 
399         // ==== Check if network already exists ====
400         networkAsBytes, err := stub.GetState(networkName)
401         if err != nil {
402                 return shim.Error("Failed to get network: " + err.Error())
403         } else if networkAsBytes != nil {
404                 fmt.Println("This network already exists: " + networkName)
```

```
405                     return shim.Error("This network already exists: " + networkName)
406             }
407
408             // ==== Create network object and marshal to JSON ====
409             objectType := "network"
410             var transcount int
411             transcount = 0
412             network := &network{objectType, networkName, address, hosts, availhosts, bandwidth, balance,
                 ↪   pricepermb, tennantslice, asnum, asneighbor, transcount}
413             networkJSONasBytes, err := json.Marshal(network)
414             if err != nil {
415                     return shim.Error(err.Error())
416             }
417
418             // === Save network to state ===
419             err = stub.PutState(networkName, networkJSONasBytes)
420             if err != nil {
421                     return shim.Error(err.Error())
422             }
423
424             //  ==== Index the network to enable avaliable hosts-based range queries,
425             //  e.g. return all networks with avaliable hosts. ==== An 'index' is a normal
426             //  key/value entry in state. The key is a composite key, with the elements
427             //  that you want to range query on listed first. In our case, the composite
428             //  key is based on indexName~availhosts~name. This will enable very efficient
429             //  state range queries based on composite keys matching
430             //  indexName~availhosts~* ====
431             asstr := strconv.Itoa(network.ASnum)
432             ASneighborasBytes, err := json.Marshal(network.ASneighbor)
433             if err != nil {
434                     return shim.Error(err.Error())
435             }
436             ASneighborasString := string(ASneighborasBytes)
437
438             indexName := "ASnum~ASneighbor"
439             availhostsNameIndexKey, err := stub.CreateCompositeKey(indexName, []string{asstr,
                 ↪   ASneighborasString})
440             if err != nil {
441                     return shim.Error(err.Error())
442             }
443             //  Save index entry to state. Only the key name is needed, no need to store
444             //  a duplicate copy of the network. Note - passing a 'nil' value will
445             //  effectively delete the key from state, therefore we pass null character as value
446             value := []byte{0x00}
447             stub.PutState(availhostsNameIndexKey, value)
448
449             // ==== Network saved and indexed. Return success ====
450             fmt.Println("- end init network")
451             return shim.Success(nil)
452     }
453
454     // ===========================================
```

```
455 // readNetwork - read a network from chaincode state
456 // ============================================
457 func (t *SimpleChaincode) readNetwork(stub shim.ChaincodeStubInterface, args []string) pb.Response {
458         var name, jsonResp string
459         var err error
460 
461         if len(args) != 1 {
462                 return shim.Error("Incorrect number of arguments. Expecting name of the network to
                    ↪  query")
463         }
464 
465         name = args[0]
466         valAsbytes, err := stub.GetState(name) //get the network from chaincode state
467         if err != nil {
468                 jsonResp = "{\"Error\":\"Failed to get state for " + name + "\"}"
469                 return shim.Error(jsonResp)stub.GetHistoryForKey(networkName)
470         } else if valAsbytes == nil {
471                 jsonResp = "{\"Error\":\"Network does not exist: " + name + "\"}"
472                 return shim.Error(jsonResp)
473         }
474 
475         return shim.Success(valAsbytes)
476 }
477 
478 // ============================================
479 // readNetwork - Returns the shortest path from AS1 to AS2.
480 // ============================================
481 func (t *SimpleChaincode) ASPath(stub shim.ChaincodeStubInterface, args []string) pb.Response {
482         //var name, jsonResp string
483         var err error
484 
485         if len(args) != 2 {
486                 return shim.Error("Incorrect number of arguments. Expecting 2 network names to query")
487         }
488 
489         //   0              1
490         // "network1", "network2"
491         if len(args) != 5 {
492                 return shim.Error("Incorrect number of arguments. Expecting 5")
493         }
494 
495         // ==== Input sanitation ====
496         fmt.Println("- start aspath network")
497         if len(args[0]) <= 0 {
498                 return shim.Error("1st argument must be a non-empty string")
499         }
500         if len(args[1]) <= 0 {
501                 return shim.Error("2nd argument must be a non-empty string")
502         }
503 
504         network1Name := args[0]
505 
```

```
506         network2Name := args[1]
507
508         var jsonResp string
509         var network1JSON network
510         var network2JSON network
511
512         valAsbytes, err := stub.GetState(network1Name) //get the network from chaincode state
513         if err != nil {
514                 jsonResp = "{\"Error\":\"Failed to get state for " + network1Name + "\"}"
515                 return shim.Error(jsonResp)
516         } else if valAsbytes == nil {
517                 jsonResp = "{\"Error\":\"Network does not exist: " + network1Name + "\"}"
518                 return shim.Error(jsonResp)
519         }
520
521         err = json.Unmarshal([]byte(valAsbytes), &network1JSON)
522         if err != nil {
523                 jsonResp = "{\"Error\":\"Failed to decode JSON of: " + network1Name + "\"}"
524                 return shim.Error(jsonResp)
525         }
526
527         valAsbytes2, err := stub.GetState(network2Name) //get the network from chaincode state
528         if err != nil {
529                 jsonResp = "{\"Error\":\"Failed to get state for " + network2Name + "\"}"
530                 return shim.Error(jsonResp)
531         } else if valAsbytes == nil {
532                 jsonResp = "{\"Error\":\"Network does not exist: " + network2Name + "\"}"
533                 return shim.Error(jsonResp)
534         }
535         err = json.Unmarshal([]byte(valAsbytes2), &network2JSON)
536         if err != nil {
537                 jsonResp = "{\"Error\":\"Failed to decode JSON of: " + network2Name + "\"}"
538                 return shim.Error(jsonResp)
539         }
540
541         return shim.Success(nil)
542 }
543
544 // ============================================
545 // delete - remove a network and its data from the state database
546 // ============================================
547 func (t *SimpleChaincode) delete(stub shim.ChaincodeStubInterface, args []string) pb.Response {
548         var jsonResp string
549         var networkJSON network
550         if len(args) != 1 {
551                 return shim.Error("Incorrect number of arguments. Expecting 1")
552         }
553         networkName := args[0]
554         // to maintain the availhosts~name index, we need to read the network first and get its
          ↪  availhosts
555         valAsbytes, err := stub.GetState(networkName) //get the network from chaincode state
556         if err != nil {
```

```
557                 jsonResp = "{\"Error\":\"Failed to get state for " + networkName + "\"}"
558                 return shim.Error(jsonResp)
559         } else if valAsbytes == nil {
560                 jsonResp = "{\"Error\":\"Network does not exist: " + networkName + "\"}"
561                 return shim.Error(jsonResp)
562         }
563
564         err = json.Unmarshal([]byte(valAsbytes), &networkJSON)
565         if err != nil {
566                 jsonResp = "{\"Error\":\"Failed to decode JSON of: " + networkName + "\"}"
567                 return shim.Error(jsonResp)
568         }
569
570         err = stub.DelState(networkName) //remove the nework from chaincode state
571         if err != nil {
572                 return shim.Error("Failed to delete state:" + err.Error())
573         }
574
575         // maintain the index
576         indexName := "availhosts~name"
577         availhostsNameIndexKey, err := stub.CreateCompositeKey(indexName,
          ↪   []string{networkJSON.Availhosts, networkJSON.Name})
578         if err != nil {
579                 return shim.Error(err.Error())
580         }
581
582         //  Delete index entry to state.
583         err = stub.DelState(availhostsNameIndexKey)
584         if err != nil {
585                 return shim.Error("Failed to delete state:" + err.Error())
586         }
587
588         return shim.Success(nil)
589 }
590
591 // ===============================================
592 // queryAvailhosts - finds a network by name and returns its current state
593 // ===============================================
594 func (t *SimpleChaincode) queryAvailhosts(stub shim.ChaincodeStubInterface, args []string) pb.Response
     ↪   {
595
596         //   0
597         // "bob"
598         if len(args) < 1 {
599                 return shim.Error("Incorrect number of arguments. Expecting 1")
600         }
601
602         name := strings.ToLower(args[0])
603
604         queryString := fmt.Sprintf("{\"selector\":{\"docType\":\"network\",\"availhosts\":\"%s\"}}",
          ↪   name)
605
```

```
606         queryResults, err := getQueryResultForQueryString(stub, queryString)
607         if err != nil {
608                 return shim.Error(err.Error())
609         }
610         return shim.Success(queryResults)
611 }
612
613 // ===================================================
614 // queryBandwidth - finds a network above, below or equal to a bandwidth value.
615 // ===================================================
616 func (t *SimpleChaincode) queryBandwidth(stub shim.ChaincodeStubInterface, args []string) pb.Response {
617
618         //   0
619         // "bob"
620         if len(args) < 2 {
621                 return shim.Error("Incorrect number of arguments. Expecting 2 ex. \"gt\",\"4000\".  Can
                  ↪   use \"lt\" - less than, \"eq\" - equal, \"ne\" - not equal, \"gte\" - greater
                  ↪   than equal, \"gt\" - greater than")
622         }
623
624         operator := strings.ToLower(args[0])
625         name := strings.ToLower(args[1])
626         queryString :=
           ↪   fmt.Sprintf("{\"selector\":{\"docType\":\"network\",\"bandwidth\":{\"$%s\":%s}}}",
           ↪   operator, name)
627         queryResults, err := getQueryResultForQueryString(stub, queryString)
628         if err != nil {
629                 return shim.Error(err.Error())
630         }
631         return shim.Success(queryResults)
632 }
633
634 // ===================================================
635 // queryPricePerMB - finds a network above, below or equal to a pricepermb value.
636 // ===================================================
637 func (t *SimpleChaincode) queryPricePerMB(stub shim.ChaincodeStubInterface, args []string) pb.Response
     ↪   {
638
639         //   0
640         // "bob"
641         if len(args) < 2 {
642                 return shim.Error("Incorrect number of arguments. Expecting 2 ex. \"gt\",\"4.4\".  Can
                  ↪   use \"lt\" - less than, \"eq\" - equal, \"ne\" - not equal, \"gte\" - greater
                  ↪   than equal, \"gt\" - greater than")
643         }
644
645         operator := strings.ToLower(args[0])
646         name := args[1]
647         queryString :=
           ↪   fmt.Sprintf("{\"selector\":{\"docType\":\"network\",\"pricepermb\":{\"$%s\":%s}}}",
           ↪   operator, name)
648
```

```
649         queryResults, err := getQueryResultForQueryString(stub, queryString)
650         if err != nil {
651                 return shim.Error(err.Error())
652         }
653         return shim.Success(queryResults)
654 }
655
656 // ===============================================
657 // queryPricePerMB - finds a network above, below or equal to a host value.
658 // ===============================================
659 func (t *SimpleChaincode) queryHosts(stub shim.ChaincodeStubInterface, args []string) pb.Response {
660
661         //   0
662         // "bob"
663         if len(args) < 2 {
664                 return shim.Error("Incorrect number of arguments. Expecting 2 ex. \"gt\",\"35\".  Can
                  ↪   use \"lt\" - less than, \"eq\" - equal, \"ne\" - not equal, \"gte\" - greater
                  ↪   than equal, \"gt\" - greater than")
665         }
666
667         operator := strings.ToLower(args[0])
668         name := strings.ToLower(args[1])
669         queryString := fmt.Sprintf("{\"selector\":{\"docType\":\"network\",\"hosts\":{\"$%s\":%s}}}",
             ↪   operator, name)
670
671         queryResults, err := getQueryResultForQueryString(stub, queryString)
672         if err != nil {
673                 return shim.Error(err.Error())
674         }
675         return shim.Success(queryResults)
676 }
677
678 // ===============================================
679 // queryPricePerMB - finds a network above, below or equal to a host value.
680 // ===============================================
681 func (t *SimpleChaincode) queryNetwork(stub shim.ChaincodeStubInterface, args []string) pb.Response {
682
683         //   0
684         // "queryString"
685         if len(args) < 1 {
686                 return shim.Error("Incorrect number of arguments. Expecting 1")
687         }
688
689         queryString := args[0]
690         //DELETE//return shim.Success([]byte(queryString))
691         queryResults, err := getQueryResultForQueryString(stub, queryString)
692         if err != nil {
693                 return shim.Error(err.Error())
694         }
695         return shim.Success(queryResults)
696 }
697
```

```
698 // ===================================================
699 // getQueryResultForQueryString executes the passed in query string.
700 // Result set is built and returned as a byte array containing the JSON results.
701 // ===================================================
702 func getQueryResultForQueryString(stub shim.ChaincodeStubInterface, queryString string) ([]byte, error)
     ↪  {
703
704         fmt.Printf("- getQueryResultForQueryString queryString:\n%s\n", queryString)
705         resultsIterator, err := stub.GetQueryResult(queryString)
706         if err != nil {
707                 return nil, err
708         }
709         defer resultsIterator.Close()
710
711         // buffer is a JSON array containing QueryRecords
712         var buffer bytes.Buffer
713         buffer.WriteString("[")
714
715         bArrayMemberAlreadyWritten := false
716         for resultsIterator.HasNext() {
717                 queryResponse, err := resultsIterator.Next()
718                 if err != nil {
719                         return nil, err
720                 }
721                 // Add a comma before array members, suppress it for the first array member
722                 if bArrayMemberAlreadyWritten == true {
723                         buffer.WriteString(",")
724                 }
725                 buffer.WriteString("{\"Key\":")
726                 buffer.WriteString("\"")
727                 buffer.WriteString(queryResponse.Key)
728                 buffer.WriteString("\"")
729
730                 buffer.WriteString(", \"Record\":")
731                 // Record is a JSON object, so we write as-is
732                 buffer.WriteString(string(queryResponse.Value))
733                 buffer.WriteString("}")
734                 bArrayMemberAlreadyWritten = true
735         }
736         buffer.WriteString("]")
737
738         fmt.Printf("- getQueryResultForQueryString queryResult:\n%s\n", buffer.String())
739
740         return buffer.Bytes(), nil
741 }
742
743 // ===================================================
744 // getHistoryForNetwork -Reads the blockchain for the history of the given key.
745 // ===================================================
746 func (t *SimpleChaincode) getHistoryForNetwork(stub shim.ChaincodeStubInterface, args []string)
     ↪  pb.Response {
747
```

```
748         if len(args) < 1 {
749                 return shim.Error("Incorrect number of arguments. Expecting 1")
750         }
751
752         networkName := args[0]
753
754         fmt.Printf("- start getHistoryForNetwork: %s\n", networkName)
755
756         resultsIterator, err := stub.GetHistoryForKey(networkName)
757         if err != nil {
758                 return shim.Error(err.Error())
759         }
760         defer resultsIterator.Close()
761
762         // buffer is a JSON array containing historic values for the network1
763         var buffer bytes.Buffer
764         buffer.WriteString("[")
765
766         bArrayMemberAlreadyWritten := false
767         for resultsIterator.HasNext() {
768                 response, err := resultsIterator.Next()
769                 if err != nil {
770                         return shim.Error(err.Error())
771                 }
772                 // Add a comma before array members, suppress it for the first array member
773                 if bArrayMemberAlreadyWritten == true {
774                         buffer.WriteString(",")
775                 }
776                 buffer.WriteString("{\"TxId\":")
777                 buffer.WriteString("\"")
778                 buffer.WriteString(response.TxId)
779                 buffer.WriteString("\"")
780
781                 buffer.WriteString(", \"Value\":")
782                 // if it was a delete operation on given key, then we need to set the
783                 //corresponding value null. Else, we will write the response.
784                 if response.IsDelete {
785                         buffer.WriteString("null")
786                 } else {
787                         buffer.WriteString(string(response.Value))
788                 }
789
790                 buffer.WriteString(", \"Timestamp\":")
791                 buffer.WriteString("\"")
792                 buffer.WriteString(time.Unix(response.Timestamp.Seconds,
                  ↪   int64(response.Timestamp.Nanos)).String())
793                 buffer.WriteString("\"")
794
795                 buffer.WriteString(", \"IsDelete\":")
796                 buffer.WriteString("\"")
797                 buffer.WriteString(strconv.FormatBool(response.IsDelete))
798                 buffer.WriteString("\"")
```

```
799
800                 buffer.WriteString("}")
801                 bArrayMemberAlreadyWritten = true
802         }
803         buffer.WriteString("]")
804
805         fmt.Printf("- getHistoryForNetwork returning:\n%s\n", buffer.String())
806
807         return shim.Success(buffer.Bytes())
808 }
```

A.7 Control Application

The control application is written in python and is used to link the Hyperledger Fabric network and the Northbound interface of the SDN controller forming a logical East/West interface between controllers.

```
1  import json
2  import subprocess
3  import re
4  import threading
5  import requests
6  import time
7
8  '''
9  ### Host Class. Store Host information after call to ONOS REST API for avaliable hosts.
10 '''
11
12
13 class Hostsclass:
14
15     def __init__(self):
16         self.mac = None
17         self.id = None
18         self.vlan = None
19         self.ipaddresses = []
20         self.locations = []
21         self.configured = bool
22         self.assigned = False
23
24     def getMac(self):
25         return self.mac
26
27     def getId(self):
28         return self.id
29
```

```
    def getVlan(self):
        return self.vlan

    def getIpaddresses(self):
        return self.ipaddresses

    def getLocations(self):
        return self.locations

    def setMac(self, arg):
        self.mac = arg
        return

    def getConfigured(self):
        return self.configured

    def getAssigned(self):
        return self.assigned

    def setId(self, arg):
        self.id = arg
        return

    def setVlan(self, arg):
        self.vlan = arg
        return

    def setIpaddresses(self, arg):
        self.ipaddresses = arg
        return

    def setLocations(self, arg):
        self.locations = arg
        return

    def setConfigured(self, arg):
        self.configured = arg
        return

    def setAssigned(self, arg):
        self.assigned = arg
        return


'''
### Node used in BST
'''


class Node:
    def __init__(self, val):
        self.val = val
```

```
82          self.leftChild = None
83          self.rightChild = None
84
85      def get(self):
86          return self.val
87
88      def set(self, val):
89          self.val = val
90
91      def getChildren(self):
92          children = []
93          if (self.leftChild != None):
94              children.append(self.leftChild)
95          if (self.rightChild != None):
96              children.append(self.rightChild)
97          return children
98
99
100 '''
101 ### BST data structure used to check if transaction has already been processed.
102 '''
103
104
105 class BST:
106     def __init__(self):
107         self.root = None
108
109     def setRoot(self, val):
110         self.root = Node(val)
111
112     def insert(self, val):
113         if (self.root is None):
114             self.setRoot(val)
115         else:
116             self.insertNode(self.root, val)
117
118     def insertNode(self, currentNode, val):
119         if (val <= currentNode.val):
120             if (currentNode.leftChild):
121                 self.insertNode(currentNode.leftChild, val)
122             else:
123                 currentNode.leftChild = Node(val)
124         elif (val > currentNode.val):
125             if (currentNode.rightChild):
126                 self.insertNode(currentNode.rightChild, val)
127             else:
128                 currentNode.rightChild = Node(val)
129
130     def find(self, val):
131         return self.findNode(self.root, val)
132
133     def findNode(self, currentNode, val):
```

```
134         if (currentNode is None):
135             return False
136         elif (val == currentNode.val):
137             return True
138         elif (val < currentNode.val):
139             return self.findNode(currentNode.leftChild, val)
140         else:
141             return self.findNode(currentNode.rightChild, val)
142 
143 
144 '''
145 ### Pretty Print JSON ###
146 '''
147 
148 
149 def pp_json(json_thing, sort=True, indents=4):
150     if type(json_thing) is str:
151         print(json.dumps(json.loads(json_thing), sort_keys=sort, indent=indents))
152     else:
153         print(json.dumps(json_thing, sort_keys=sort, indent=indents))
154     return None
155 
156 
157 '''
158 Query host from hyperledger world state.
159 '''
160 
161 
162 def queryHosts(networkName):
163     output = str(
164         subprocess.check_output("sudo docker exec cli scripts/c_queryhost.sh mychannel sdnnetwork " +
            ↪   networkName,
165                                 shell=True))
166     output = re.sub(r'.* {', '{', output).strip("\ n\n'")
167     jsonoutput = json.loads(output)
168     return jsonoutput
169 
170 
171 '''
172 Collect active hosts from ONOS.
173 '''
174 
175 
176 def collectHosts():
177     hosts = requests.get('http://172.17.0.2:8181/onos/v1/hosts', auth=('onos', 'rocks'))
178     hostList = []
179     for host in hosts.json()["hosts"]:
180         obj = Hostsclass()
181         obj.setVlan(host["vlan"])
182         obj.setLocations(host["locations"])
183         obj.setId(host["id"])
184         obj.setMac(host["mac"])
```

```
185             obj.setConfigured(host["configured"])
186             obj.setIpaddresses(host["ipAddresses"])
187             loclist = []
188             for loc in host["locations"]:
189                 loclist.append([loc["elementId"], loc["port"]])
190                 obj.setLocations(loclist)
191                 hostList.append(obj)
192                 # print(obj.getVlan(), obj.getLocations(), obj.getId(), obj.getMac(), obj.getConfigured(),
                      ↪  obj.getIpaddresses())
193         # print(hostList)
194         return hostList
195
196
197     '''
198     Performs a check of the hyperledger World State for 'network' and the Onos Hosts.
199     '''
200
201
202     def updateLists(queryHostsList, hostObjectsList, continueUpdate, network):
203         checkForHostUpdate = 50
204         checkFrequency = 40
205         while continueUpdate[0]:
206             checkForHostUpdate += 1
207             queryHostsList[0] = queryHosts(network)
208             if checkForHostUpdate > checkFrequency:
209                 hostObjectsList[0] = collectHosts()
210                 checkForHostUpdate = 0
211             time.sleep(.1)
212         return
213
214
215     '''
216     Posts an intent to ONOS. Currently ONOS Requires MAC to use the intent framework.
217     Future work to update intent framwork to work via IP address. Point to Point intents will allow for IP
      ↪  address.
218     '''
219
220
221     def postintents(host, externalHost):
222         intentDict = {1: "HostToHostIntent", 2: ""}
223         print("host and externalHost:", host, externalHost)
224         hostdict = {}
225         print(host.getId())
226         hostdict["type"] = intentDict[1]
227         hostdict["table"] = 1
228         hostdict["appId"] = "org.onosproject.ovsdb"
229         hostdict["one"] = externalHost
230         hostdict["two"] = "00:00:00:00:00:02/None"  #hardcoded for testing
231         #hostdict["two"] = host.getMac()
232         intentpost = json.dumps(hostdict)
233         pp_json(intentpost)
```

```
234     request_posts = requests.post('http://172.17.0.2:8181/onos/v1/intents', auth=('onos', 'rocks'),
          ↪   data=intentpost)
235     print("POST response;", request_posts)
236     return
237
238
239 if __name__ == "__main__":
240     network = "network1"
241     gatewaymac = "08:00:27:9B:77:37/None"
242     supportedTennants = {}
243     queryHostsList = [[]]
244     hostObjectsList = [{}]
245     continueUpdate = [True]
246     bst = BST()
247     t = threading.Thread(target=updateLists, args=(queryHostsList, hostObjectsList, continueUpdate,
          ↪   network))
248     t.start()
249     time.sleep(5)
250     while True:
251         time.sleep(.3)
252         for tenant in queryHostsList[0]["tenantslice"]:
253             if not bst.find(tenant["transnum"]):
254                 bst.insert(tenant["transnum"])
255                 print("INSERTED")
256                 count = tenant["hosts"]
257                 current_count = 0
258                 for host in hostObjectsList[0]:
259                     if not host.getAssigned():
260                         host.setAssigned(True)
261                         for address in tenant["saddress"]:
262                             current_count += 1
263                             if current_count <= count:
264                                 postintents(host, gatewaymac)
265                             else:
266                                 break
267
268
269     #continueUpdate[0] = False
270     #t.join()
```

List of References

[1] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.

[2] S. Jain, A. Kumar, S. M, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. W, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, and G. Inc, "B4: Experience with a globally-deployed software defined wan."

[3] Noxrepo, "noxrepo/nox," Feb 2014. Available: https://github.com/noxrepo/nox

[4] "Home." Available: https://www.opendaylight.org/

[5] "Open network operating system." Available: https://onosproject.org/

[6] "Download." Available: https://osrg.github.io/ryu/

[7] Available: https://openflow.stanford.edu/display/Beacon/Home.html

[8] "Floodlight openflow controller -." Available: http://www.projectfloodlight.org/floodlight/

[9] "Opencontrail is an open source network virtualization platform for the cloud." Available: http://www.opencontrail.org/

[10] F. Le, C. Leet, C. Makaya, M. Rio, X. Wang+, and Y. R. Yang, "Sfp: Toward a scalable, efficient, stable protocol for federation of software defined networks," in *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug 2017, pp. 1–6.

[11] S. Raju, S. Boddepalli, S. Gampa, Q. Yan, and J. S. Deogun, "Identity management using blockchain for cognitive cellular networks," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[12] T. Parth, N. Senthil, and V. Balaji, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," *arXiv:1805.11390*, May 2018.

[13] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains," Internet Requests for Comments, Internet Research Task Force, RFC, December 2012. Available: https://tools.ietf.org/html/draft-yin-sdn-sdni-00

[14] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "Sdx: A software defined internet exchange," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 551–562, Aug. 2014. Available: http://doi.acm.org/10.1145/2740070.2626300

[15] P. Goransson, C. Black, and T. Culver, *Software Defined Networks*, 2nd ed. Burlington, MA: Morgan Kaufmann, 2016.

[16] B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. Talayco, D. Erickson, D. McDysan, D. Ward, E. Crabbe, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Tonsing, J. Pettit, K. Yap, L. Poutievski, L. Vicisano, M. Casado, M. Takahashi, M. Kobayashi, N. Yadav, N. McKeown, N. dHeureuse, P. Balland, R. Ramanathan, R. Price, R. Sherwood, S. Das, S. Gandham, T. Yabe, Y. Yiakoumis, and Z. L. Kis., "Openflow switch specification version 1.3.0 (wire protocol 0x04)," no. ONF TS-006, pp. 1–106, June 2012. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[17] Y. I. Daradkeh, M. Aldhaifallah, D. Namiot, and M. Sneps-Sneppe, "On standards for application level interfaces in sdn," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 10, 2016. Available: http://dx.doi.org/10.14569/IJACSA.2016.071006

[18] "Open vswitch documentation¶." Available: http://docs.openvswitch.org/en/latest/

[19] Mininet, "mininet/mininet." Available: https://github.com/mininet/mininet/blob/master/examples/controllers2.py

[20] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, pp. 99–111, 1991.

[21] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. Available: http://www.bitcoin.org/bitcoin.pdf

[22] R. Caetano, *Learning Bitcoin*. Packt Publishing, 2015.

[23] "Bitcoin.com." Available: https://charts.bitcoin.com/btc/chart/blockchain-size

[24] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, June 2014, pp. 280–285.

[25] "data.bitcoinity.org (beta version)." Available: https://data.bitcoinity.org/bitcoin/difficulty/5y?t=l

[26] V. Buterin, "A next generation smart contract decentralized application platform," 2013.

[27] etherscan.io, "Charts ethereum chaindata size growth (fast sync)." Available: https://etherscan.io/chart2/chaindatasizefast

[28] Hyperledger, "Hyperledger: Introduction¶." Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html

[29] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *CoRR*, vol. abs/1801.10228, 2018. Available: http://arxiv.org/abs/1801.10228

[30] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, "Xft: Practical fault tolerance beyond crashes," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (OSDI'16). Berkeley, CA, USA: USENIX Association, 2016, pp. 485–500. Available: http://dl.acm.org/citation.cfm?id=3026877.3026915

[31] R. Mahajan, D. Wetherall, and T. Anderson, "Towards coordinated interdomain traffic engineering," *Proceedings of Third Workshop on Hot Topics in Networks (HotNets-III)*, August 2004.

[32] M. Howard and H. Adams, "Operator survey: Smart central offices to be in 85 percent of service provider networks this year - ihs technology," Jan 2018. Available: https://technology.ihs.com/599622/operator-survey-smart-central-offices-to-be-in-85-percent-of-service-provider-networks-this-year

[33] Mininet. (2018). Mininet homepage. [Online]. Available: www.mininet.org. Accessed July 14, 2018.

[34] ONOS. (2018). ONOS Wiki homepage. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Wiki+Home. Accessed July 14, 2018.

[35] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," *Passive and Active Measurement Lecture Notes in Computer Science*, p. 31–41, 2013.

[36] Hyperledger, "hyperledger/education," Oct 2017. Available: https://github.com/hyperledger/education/tree/master/LFS171x/fabric-material/basic-network

[37] “Membership service providers (msp).” Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/msp.html

[38] “Couchdb as the state database.” Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/couchdb_as_state_database.html

[39] “high-throughput/scripts · master · khanhtran / fabric-server.” Available: https://git.syncfab.com/khanhtran/fabric-server/tree/master/high-throughput/scripts

[40] “Advanced traffic control.” Available: https://wiki.archlinux.org/index.php/Advanced_traffic_control

[41] A. Ledenev, “alexei-led/pumba,” Nov 2018. Available: https://github.com/alexei-led/pumba

[42] Opennetworkinglab, “opennetworkinglab/onos.” Available: https://github.com/opennetworkinglab/onos/blob/master/tools/test/topos/default.py

[43] Opennetworkinglab, “opennetworkinglab/onos.” Available: https://github.com/opennetworkinglab/onos/blob/master/tools/test/topos/onosnet.py

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California