QATAR UNIVERSITY

COLLEGE OF ENGINEERING

ENSURING QUALITY OF SERVICE IN BLOCKCHAIN-BASED HEALTHCARE SYSTEM

BY

SALMA T. SHALABY

A Thesis Submitted to

the College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Masters of Science in Computing

June 2020

COMMITTEE PAGE

The members of the Committee approve the Thesis of
Salma Shalaby defended on 22/04/2020.

Dr. Abdulla Al-Ali
Thesis/Dissertation Supervisor

Dr. Amr Mohammed
Co-Supervisor

Dr. Mohamed Abdallah
Committee Member

Dr. Aiman Erbad
Committee Member

Dr. Nizar Zorba
Committee Member

Approved:

Khalid Kamal Naji, Dean, College of Engineering

ABSTRACT

SHALABY, SALMA, T., Masters : June : [2020], Masters of Science in Computing

Title: Ensuring Quality of Service in Blockchain-Based Healthcare System

Supervisor of Thesis: Abdulla, K., Al-Ali and Amr, M., Mohammed.

Blockchain is a distributed secure ledger that eliminates the need for centralized authority to store data. It provides decentralized, secure and trustless framework that does not require a third party for transaction processing, while enhancing fault tolerance. In this thesis, we investigate the potentials of customizing the behavior of Blockchain network for versatile healthcare applications' requirements. Firstly, we conduct several experiments to evaluate the performance of the Hyperledger Fabric (HLF) – a permissioned Blockchain framework. Several scenarios were evaluated to depict the Blockchain behavior in terms of end-to-end transaction latency and network throughput. In the second phase, we leverage a Blockchain framework that provides Quality of Service (QoS) by integrating it with smart health system where the edge gateway decides how the data will be sent to the Blockchain network by prioritizing the transactions based on the patient's case using the notion of Blockchain channels. We design a system with three-channel Blockchain network, each has different configuration, which enables some transactions to be processed faster than the others. The results show that channels can be configured to provide fast track with minimal latency regardless of the frequency of the transactions, which guarantees that urgent transactions will have highest priority. On the other hand, other channels' performance varies depending on the number of transactions received and the frequency of sending them.

DEDICATION

To my Parents,

for their endless love and support

ACKNOWLEDGMENTS

First, and foremost I praise Allah for giving me the power and guiding me through this journey. Without his guidance nothing could have been accomplished, Alhamdullah.

Moreover, I would like to express my deep sincere and gratitude to my supervisors Dr. Amr and Dr. Abdulla for their guidance and their valuable feedback throughout this work. A Special thanks to Dr. Alaa Awad for his collaboration and for his valuable advices. I also want to thank Eng. Tam for helping me in the implementation at some point of this work.

Lastly, I would like to thank my family and my friends for their continuous love, support and encouragement, and for having faith in me.

TABLE OF CONTENTS

DEDICATION iv

ACKNOWLEDGMENTS v

LIST OF TABLES ix

LIST OF FIGURES x

Chapter 1: Introduction 1

1. Motivation 1
2. Research Questions 2
3. Contribution 2
4. Document Overview 3

Chapter 2: Background 4

1. Blockchain 4
   - 1.1. Permissioned and Permission-less Blockchain 5
   - 1.2. Permissioned Blockchain 7
2. Edge Computing 16
   - 2.1. What is Edge Computing? 16
   - 2.2. Benefits of Edge Computing 17
   - 2.3. Integrating Edge Computing and Blockchain 18
3. Literature Review 22

Chapter 3: Performance Evaluation of HLF 26

1. Building the Network 26

1.1. Network Infrastructure 26

1.2. Application Layer 27

2. Experiments 29

3. Results 31

3.1. Batch-timeout 31

3.2. Batch Size (Max. Number of Transactions per Block) 32

3.3. Endorsing peers 34

Chapter 4: Ensuring QoS on Blockchain 36

1. System Architecture 36

1.1. Multi-channel Blockchain framework 37

1.2. Edge Computing 39

2. Experiments 40

3. Results and Discussion 41

3.1. Fixed Time Gap 41

3.2. Varying Time Gap 43

3.3. Varying the Transactions Frequency 44

Chapter 5: Conclusion and Future Work 48

1. Conclusion 48

2. Future Work 49

References..50

Appendix A: Multi-Channel Blockchain Implementation Steps.................................55

LIST OF TABLES

Table 1: Emergeny Implication of Signal States 19

Table 2: Association Rules Generated from TD Features [32]................................... 21

Table 3: The Business Model 28

Table 4: Experimental Setup.. 30

Table 5: System Specifications – Phase 1... 31

Table 6: Matching Channels to Transactions' Emergency.. 37

Table 7: Channels Configuration... 40

Table 8: System Specifications – Phase 2... 41

Table 9: Experiments 1 results (Fixed time Gap).. 42

Table 10: Experiments 2 Results (Varying time Gap).. 44

LIST OF FIGURES

Figure 1. Simple representation of blockchain building blocks 4

Figure 2: Channels Concept 11

Figure 3: Execute-order-validate 13

Figure 4: Proposal phase 14

Figure 5: Ordering and packaging phase 15

Figure 6: Validation and committing phase 16

Figure 7: Rule extraction and classification Steps [32] 19

Figure 8: Converting TD feature values to fuzzy relation [32] 20

Figure 9: Converting TD fuzzy to crisp relation and finding optimal rectangle [32] .. 21

Figure 10: Block structure in a) Private blockchain b) Consortium blockchain [37] .. 23

Figure 11: High level architecture of the network 26

Figure 12: Average latency for varying batch-timeout 32

Figure 13: Average throughput for varying batch-timeout 32

Figure 14: Average latency for varying batch Size 33

Figure 15: Number of blocks commited 34

Figure 16: Average throughput with varying batch size 34

Figure 17: Average latency for varying number of endorsers 35

Figure 18: System architecture 36

Figure 19: Multi-channel blockchain framework 37

Figure 20: Deciding the transmission channel at the edge gateway 39

Figure 21: Transactions latency with fixed time gap 43

Figure 22: Transactions latency with varying time gap 44

Figure 23: Effect of varying the transactions frequency 46

Figure 24: Number of transactions per block 46

Figure 25: Generating the Cryptographic material of the organizations 55

Figure 26: Creating Raft Genesis Block 55

Figure 27: Channel 1 artifacts creation and defining anchor peers 56

Figure 28: Channel 2 artifacts creation and defining anchor peers 56

Figure 29: Channel 3 artifacts creation and defining anchor peers 56

Figure 30: Bringing the network up..57

Figure 31: Creating channel-1 block and peers join the channel.................................57

Figure 32: Updating anchor peers' definition inn channel 1 58

Figure 33: Creating channel 2 block and peers join the channel 58

Figure 34: Updating anchor peers' definition in channel 2 58

Figure 35: Creating channel 3 block and peers join the channel 59

Figure 36: Updating anchor peers' definition in channel 3 59

Figure 37: Installing the chaincode on the peers 59

Figure 38: Instantiating the chaincode on channel 1 60

Figure 39: Instantiating the chaincode on channel 2 60

Figure 40: Instantiating the chaincode on channel 3 60

Figure 41: Fetching the current configuration of channel 3 60

Figure 42: Convert the portobuf format to readable JSON format..............................60

Figure 43: create a new copy of the config file 60

Figure 44: Applying the modifications..61

Figure 45: converting teh modified JSON file to portobuf format..............................61

Figure 46: Computing teh difference and converting it to JSON 61

Figure 47: Signing the updates and sending the update transaction 62

CHAPTER 1: INTRODUCTION

1. Motivation

In 2009, Blockchain emerged as a distributed ledger for bitcoin transactions. Although it was first introduced for cryptocurrency transactions, the financial service is not the only application where Blockchain can be utilized. It can be used in different sectors such as: business, industry, healthcare applications, Internet of Things (IoT) and much more [1]. Blockchain is a distributed database that consists of chained blocks that store the data. Each of these blocks, except the first block, is linked to the hash of the previous block, which ensures that any change in the data will be recognized [2].

One of the important aspects that should be considered while sharing healthcare data is preserving privacy. Healthcare data gains its sensitivity from the fact that it is related to patients' privacy and sharing will require a secure environment. Hence, there is a need for a decentralized ledger that shares the data securely between different entities, i.e. hospitals, Ministry of health, patients, etc. Compared to the centralized approach, the decentralized systems do not require a man-in-middle to monitor and facilitate the communication. Additionally, it is not subject to single point of failure [3].

Sharing and managing healthcare information is crucial because it provides a full view of the patient's state and engages him/her in the treatment process. It also helps in tracking the health trends within a country. In addition, this data affects the business decisions. Healthcare data is not only valuable for the hospital or the medical organization that is issuing it. It is also used by other entities like the ministry of health for statistics, research centers and universities for doing researches, patients and other entities that might need the data, and insurance companies to provide premium healthcare services. Hence, there is a need for these versatile health organizations to have a peer-to-peer trust (i.e. preferably with no mediator to preserve privacy) in order to facilitate the efficient exchange of medical information. However, one of the main concerns in sharing healthcare data is the patient's privacy, the personal data of the patient and his/her medical records are valuable for attackers for different reasons. Hence, to guarantee that this data cannot be accessed or tampered by any unauthorized user, the sharing environment should be completely secure and efficient. Another aspect is the diversity of policies and decision-making processes for the medical stakeholders. For example, entities such as health ministry, hospitals, drug organizations, etc. will have diverse policies that makes the exchange of medical data a real challenge. Therefore, blockchain platform will work on sharing the data between the different entities with providing different access levels based on using digital smart contracts that will help insure all policies are validated for different access level of the medical data exchange process. Besides the privacy of the patients, one of the points that should be considered while sharing medical data is the priority; some data has high emergency level, which requires minimal delay while some information can tolerate latency. Thus, Quality of Service (QoS) should be considered while sharing medical data.

2. Research Questions

- How can the block size and batch-timeout parameters affect the blockchain performance in terms of end-to-end latency and throughput?
- How can Blockchain help in providing QoS in transferring medical data?

3. Contribution

1. Building a secure multi-channel Blockchain framework with customized smart contract for sharing healthcare data
2. Studying the behavior of Hyperledger Fabric (HLF) and evaluating its performance by conducting several experiments that shows the effect of different parameters on the overall performance
3. Integrating edge computing with our multi-channel blockchain framework to

prioritize the transactions based on their urgency and ensure QoS

4. Document Overview

The rest of this document is divided as follows:

- **Chapter 2**: Provides a background of Blockchain and edge computing in addition to the literature review
- **Chapter 3**: Performance evaluation of HLF that studies how different parameters can affect the performance
- **Chapter 4**: Integration of edge computing with multi-channel blockchain framework to ensure QoS
- **Chapter 5**: Conclusion and future work

CHAPTER 2: BACKGROUND

This chapter introduces the main concepts used in this work. It starts with a background about Blockchain and edge computing, then, it studies the previous work done in the area of using Blockchain in healthcare applications.

1. Blockchain

In 2009 Bitcoin emerged as the first decentralized cryptocurrency [4]. When Satoshi Nakamoto came up with this new digitalized currency, the main aim was to create a system that is not controlled by a single entity (i.e. banks). This was achieved by replacing the centralized database by Blockchain, which is a distributed ledger that could be accessed by everyone. As the name implies, Blockchain consists of connected blocks such that each block is attached to the previous one by storing its hash as shown in fig. 1. The fact that each block stores the hash of the previous block guarantees integrity, because if the data changed in any block its hash will change, hence, it will not match the previous hash in the next block anymore and the change will propagate through the rest of the chain and it will be recognized. Moreover, Blockchain enables the usage of smart contracts which is an automated program that controls the transaction logic and does not require an intermediary to run it [5].

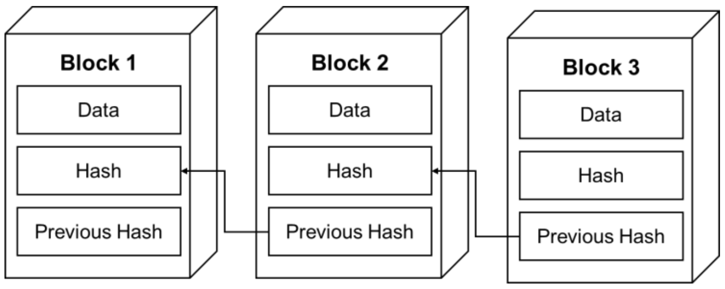


Figure 1. Simple representation of blockchain building blocks

1.1. Permissioned and Permission-less Blockchain

When Blockchain came out, its main goal was to provide transparent transactions by having an open platform that anyone can participate in without depending on a central authority. Having such an open network is acceptable in some scenarios such as cryptocurrencies, social media, etc., however, in some applications, confidentiality and privacy are main requirements. To meet the requirements of such applications, permissioned blockchains emerged as a Distributed Ledger Technology (DLT) [6], [7]:

- **Permission-less Blockchains**: They are open to anyone to access them, interestingly, there is no need for any central authority to guarantee access to the users, which makes the network fully decentralized. However, the fact that the number of participants is high slows the network down as they all have to agree on the transactions before they are committed to the ledger
- **Permissioned Blockchains:** Only authorized users are allowed to participate in permissioned-blockchains, thus, there should be an entity or a group of entities that grant access to new participants, which results in a partially decentralized system. Compared to permission-less blockchains, permissioned blockchains are faster because the number of participants is usually much less.

Because each entity holds a copy of the ledger, they have to reach consensus between each other in order to agree on the same copy and be able to identify any unauthorized data manipulation. Proof-of-Work (PoW) and Proof-of-Stake (PoS) are two of the most popular consensus algorithms used in permission-less blockchains.

In Bitcoin for example, PoW consensus scheme is used, in which miners compete to solve a computationally intensive puzzle and once a miner solves this puzzle it broadcasts the new block, in return, the wining miner gets a reward in addition to the transactions' fees, it is worth mentioning that this reward changes every four years in

Bitcoin Blockchain. In order to create a new block, miners compete to satisfy the following condition:

$$Hash\left(nonce\middle|\middle|data\middle|\middle|prev_{hash}\right) \leq target$$

The miner must generate a hash value that lies in the target space; in other words, the hash value generated must be less than or equal to the target. This condition can be satisfied by changing the nonce value until the miner gets a hash value that lies within this space. Miners compete to satisfy this condition and the first miner to solve this puzzle broadcasts the block and gets rewarded, if the block got verified. Although this puzzle is hard to solve, it can be easily verified by other miners using the same input that the broadcasting miner provided and generating the hash value. The difficulty of this process comes from the fact that the target space is much smaller than the hash output space given that Bitcoin uses SHA256 hashing algorithm which generates a hash of 256 bits. Bitcoin Blockchain readjusts the target every 2016 blocks to guarantee that one block is generated every 10 minutes in average; if the average time for generating the blocks was less than 10 minutes, the target value decreases, which makes the puzzle harder because the target space becomes smaller. In contrast, if the average time was found to be more than 10 minutes, the target value is readjusted to a higher value giving a bigger space to ease the mining process [8]. One of the limitations of PoW is that it is vulnerable to 51% attack, which can happen if a single entity owns more than 51% of the computational power of the Blockchain enabling them to take control of the whole network [9].

Peercoin [10] proposed PoS to decrease the computational overhead of PoW. PoS depends on the amount of time that the miner holds a certain amount of currency. This approach uses coin-age that is the number of days of holding the currency, times the amount that it has been held. Based on the coin-age, the miners are chosen; the probability of being chosen as the next block miner increases as the coin-age increases, until it is maximized when the coin-age is 90 days. Unlike PoW, this approach does not consume huge amount of resources. Also, it is not vulnerable to 51% attack as the attacker needs to own more coins than the rest of the network; causing an increase in the coin price, which makes the attack very costly and almost impossible [11]. PoW and PoS are two of the most common consensus techniques in permission-less blockchains that guarantee trust, however the mining process is time consuming. On the contrary, permissioned Blockchain leverages faster protocols to achieve consensus [12].

1.2.Permissioned Blockchain

Permissioned blockchains showed their ability to provide the confidentiality requirements needed by some of applications and business use cases, they also showed better performance in terms of throughput as the consensus process is not as slow as most of permission-less consensus protocols. In this section, we explore some of the available permissioned blockchains such as: Ethereum [13], Corda [14] and MultiChain [15]. Then, Hyperledger Fabric [16] is explained in more details as it is the platform used in this work.

1.2.1. Ethereum

Ethereum [13] was established in 2015 and eventually it became one of the biggest programmable blockchain frameworks. It acts as permission-less blockchain to exchange Ether (ETH) cryptocurrency. However, Ethereum can offer more than this as it is programmable and open source. Thus, it is also used as a permissioned blockchain to develop customized decentralized applications [17]. While the current version of Ethereum (Eth1) relies on PoW consensus algorithm, it is planned that Eth2, which is under development, will utilize PoS [18]. Ethereum has two types of accounts [19]:

1. Externally Owned Accounts (EoA): used by the users to send their transactions to the network
2. Contract accounts: used by the smart contracts to call each other by sending internal transactions [20].

Ethereum Blockchain is considered as a state machine and the valid transactions are the events that cause the state change [21]. Initially, Ethereum transactions start from an EoA that will send ether and cause a direct change in the state, or, it can create a contract account that will call one or multiple contracts through internal transactions and finally change the state.

1.2.2. Corda

Corda [14] is an open source Blockchain designed for recording and processing business data. The main building block of Corda is the state object, which defines the ledger. This state object represents a record of the current state and content of agreement between two or more parties. Corda ledger updates are applied through transactions, which consume existing state objects and produce new ones. To reach full consensus on these transactions, two aspects are considered:

1. **Transaction Validity:** The participating parties have to check the contract code, which is required to be deterministic. Then, they have to check if the old transactions that the current transaction refers to are valid and finally, they have to check that it has all the required signatures.
2. **Transaction uniqueness:** The uniqueness can be checked by verifying that the transaction is not consuming any of the states that have been consumed earlier.

It is worth mentioning that Corda only allows the parties that are part of a transaction to participate in the transaction validation process. However, for approving the transaction uniqueness, Corda has pluggable services that allow multiple untrusting nodes to participate in this process [22].

1.2.3. MultiChain

Multichain [15] is a private blockchain that is based on a fork of Bitcoin's blockchain to serve financial institutions. The main goals of MultiChain are:

1. Provide a blockchain platform that is only accessible by chosen participants
2. Control the transactions exchanged by setting rules
3. Enable secure mining without having PoW in the loop because it is computationally expensive

In Multichain, mining is restricted to a set of identified entities, it enforces the participation of all the miners in the defined set by adopting round-robin schedule and setting the mining diversity which is the proportion of the miners who have to participate in the mining process to maintain the network. The process of validating a block is done as follows [23]:

1. Apply the changes of the assets based on the transactions
2. Calculate the spacing by multiplying the number of miners by the mining diversity. The spacing is defined as the number of blocks that each miner has to wait to be eligible to mine again
3. If the miner has mined one of the previous spacing-1 blocks, then the block is invalid.

By applying these steps, Multichain ensures that the all the identified miners have participated in the process, however, this reveals a limitation where the network might freeze if the miners are inactive especially if the mining diversity is very high. On the other hand, if the mining diversity is small then the mining process is only rotating between a small number of miners which raises the centralization level.

1.2.4. Hyperledger Fabric

Hyperledger Fabric (HLF) [16] is an open source permissioned Blockchain established by the Linux Foundation that is mainly used to serve enterprises. What distinguishes HLF from other platforms is its new transaction architecture called Execute-Order-Validate architecture. This new architecture replaces the traditional order-execute one used by all of the existing platforms. In order-execute architecture, first, transactions are ordered based on the consensus protocol. Then, in the execution phase, each peer executes transactions sequentially in the same order. This execution phase has a negative impact on the performance of the network as the peers have to go through all the transactions in the block and execute them, which increases the latency. Another point that differentiates HLF is that it supports general-purpose programming language smart contracts or "chaincodes" as HLF calls them. It is obligatory that smart contracts in platforms that follow the order-execute architecture have to be deterministic; and that is why some platforms require writing the smart contract in a Domain Specific Language (DSL) to eliminate the non-deterministic operations [24], [25].

1.2.4.1.Channels

Based on the requirements, HLF allows the creation of several channels within the same network. A channel could be referred to as a private communication network within the main network. Each channel has its own ledger, therefore, only the peers and organizations that are members of this channel will have a copy of this ledger. These members are defined in the channel policy in the configuration block that also defines the type of ordering service. Whenever any configuration is modified, a new configuration block is created and added to the chain [25]. Fig. 2 clarifies the channel concept, where organization A and organization B have access to channel 1 and organization B and organization C have access to channel 2. Peers in organization 1 have a copy of the ledger that belongs to channel 1 only. Similarly, peers in organization C have a copy of the ledger that belongs to channel 2 only. Since organization B is a member in both channels, its peers have copy of both ledgers. This could be useful in scenarios where there are competing companies as it allows private communication between the organizations on the same channel.

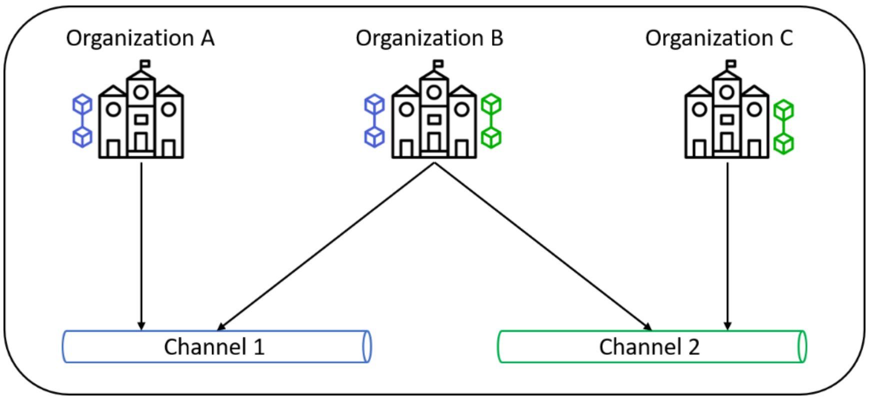


Figure 2: Channels Concept

1.2.4.2.Ordering Service

To achieve consensus, HLF introduces orderer nodes. As the name implies, the main responsibility of orderer nodes is ordering the transactions in the blocks and broadcasting them to the peers for validation. HLF offers three different implementations of the ordering service [25]:

- **Solo**: In solo-based implementation there is only one ordering node, which makes it a single-point-of-failure. Consequently, solo-based ordering service is not suitable for production. Nevertheless, it could be applied for testing and for academic usage as it eliminates the administrative overhead in the other implementation schemes.
- **Kafka**: This mode follows leader and follower approach, where the leader

orderer sends the transactions to the follower orderer nodes. The choice of the leader is done in dynamic fashion and as long as the majority of the nodes is up, the system is Crash Fault Tolerant (CFT). This scheme utilizes zookeeper coordination service [26] for managing Kafka cluster. Zookeeper helps in tasks coordination, distributed synchronization and cluster membership. Although this scheme was the only option that supports multiple orderers since HLF v1.0, the setup of Kafka-based ordering service is challenging, and it requires experts to deploy it [25].

- **Raft**: Recently, HLF added Raft ordering service that is based on Raft protocol. It is similar to Kafka, as it has the advantage of being CFT and it follows the leader and follower approach. From the functional point of view there are no differences between Raft and Kafka, however, Raft is easier to setup [25].

1.2.4.3.Transaction Flow

HLF introduces two types of transactions: *Update Transactions* and *Query Transactions*. In the former, all the peers need to agree on the transactions before updating the ledger, which is known as consensus. In order to achieve consensus in HLF, the update transactions go through a three-phase process known as *Execute-Order-Validate* process, summarized in fig. 3, the process starts once the transaction is proposed and ends when it is committed to the ledger. On the other hand, query transactions go through the first phase only because there are no changes that would affect the ledger, thus, not all the peers have to be involved [25].

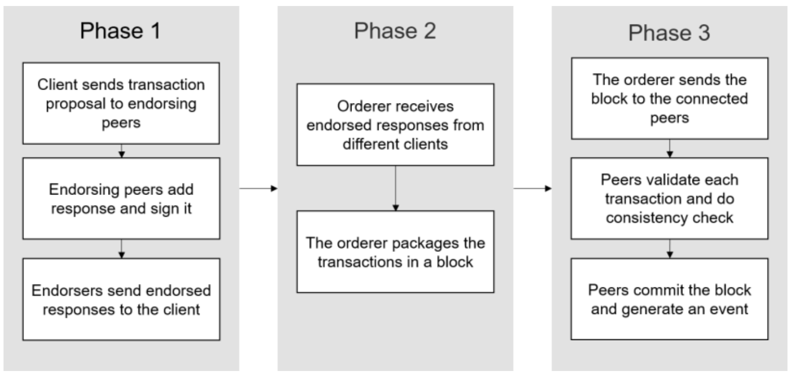

Phase 1
Client sends transaction proposal to endorsing peers
Endorsing peers add response and sign it
Endorsers send endorsed responses to the client
Phase 2
Orderer receives endorsed responses from different clients
The orderer packages the transactions in a block
Phase 3
The orderer sends the block to the connected peers
Peers validate each transaction and do consistency check
Peers commit the block and generate an event

Figure 3: Execute-order-validate

Phase 1: Proposal (Execute)

The main aim of this phase is endorsing the transactions. As shown in fig. 4, this phase is divided into 3 main steps:

1. The application client sends a transaction proposal to the endorsing peers.
    - The set of the endorsing peers chosen is determined by the endorsement policy
2. Each endorsing peer executes the chaincode individually and checks the following:
    - The proposal is well structured
    - The application is not trying to duplicate a transaction that already exists
    - The signature of the issuer is valid
    - The issuer is allowed to perform the proposed operation

   Based on the execution results the endorsing peer generates transaction response and signs it
3. The endorsers send the signed response to the client
    - Depending on the number of endorsing peers defined in the endorsement policy, the client waits until it receives a certain number of endorsements, which marks the end of the first phase.

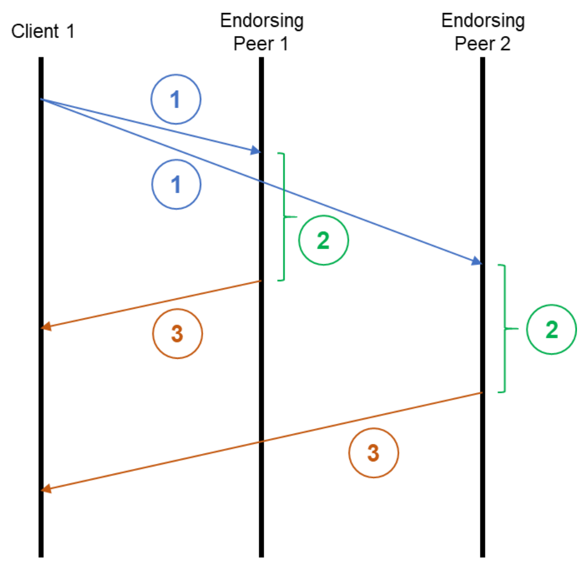


Figure 4: Proposal phase

Phase 2: Ordering and Packaging (Order)

This phase is concerned with packaging the transactions, which is the orderer's main responsibility. The orderer does not look into the content of the transactions, it only packages them into a block, unless it is a configuration transaction. As illustrated in fig. 5, this phase goes through the following steps:

1. The ordering service receives transactions response from different client applications
2. The ordering services package the transactions into a block
    - It is worth noting that the number of transactions per block is decided by the channel configuration, and it affects the overall latency of admitting the transaction into the Blockchain.

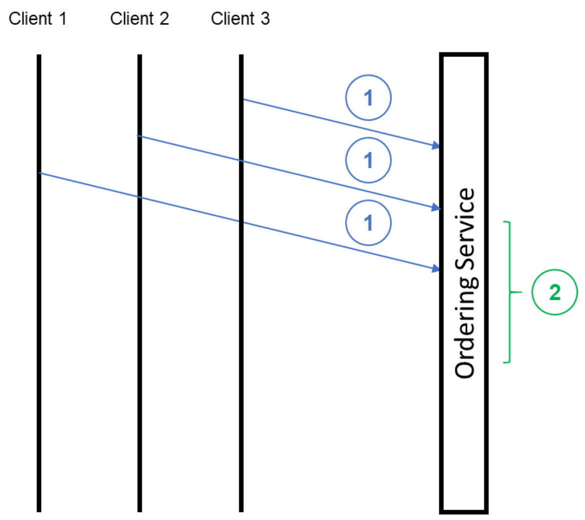


Figure 5: Ordering and packaging phase

Phase 3: Validation and Committing (Validate)

The last phase is when the block is finally committed to the ledger, it is divided into the following steps as shown in fig. 6:

1. The orderers send the block to all the peers connected to it
    - Peers that are not connected to the orderer will eventually receive the block by gossiping
2. Each peer on the channel will validate the transactions in the block separately then commit the block to the ledger
    - Although each peer validates the transactions separately, this is done in a deterministic way, which guarantees that each peer will have an identical copy of the ledger.

The process of validation includes ensuring that the transaction is endorsed by the required endorsers according to the endorsement policy. To commit the block to the ledger, the peers perform a consistency check to ensure that none of the assets was updated by any other transaction when phase 1 and 2 were taking place.

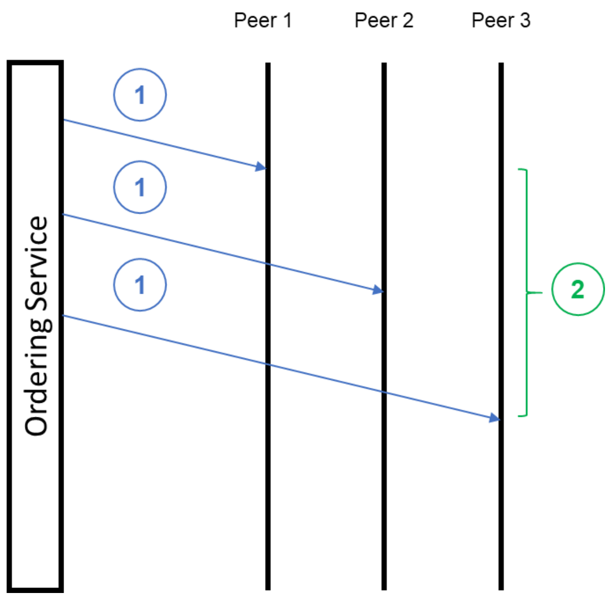


Figure 6: Validation and committing phase

2. Edge Computing

In this section, we give a background about edge computing and its benefits. Then we discuss the motivation behind integrating Blockchain and edge computing.

2.1. What is Edge Computing?

The emergence of IoT and the enormous amount of data produced by its devices changed the computing map where data was stored and processed at the cloud side and pushed towards Edge Computing [27]. The term Edge computing refers to allowing the computations to be done at the proximity of the data producers rather than doing it at the cloud side [28]. The fact that the data is generated at the edge of the network makes processing it near to the producers more efficient as it reduces the network load and consumes less time compared to cloud computing [29]. Typically, edge computing can be divided into three layers: end devices (front-end), edge server (near-end) and core cloud (far-end) [30]. The end devices are the data collectors and the actuators that interact with the environment. However, these light-weight devices cannot perform computations because of their limited resources. The edge server is introduced to facilitate traffic flow in the network and provide computational requirements such as real-time processing as well as data storage and caching. As the third tier (i.e. core cloud) is more powerful it is responsible for heavy computations and storing big data.

2.2.Benefits of Edge Computing

Applying edge computing brings several benefits that can be summarized in the following points:

1. **Faster Processing**: Although cloud computing proved to be efficient for many years, with the enormous increase of the number of devices and the amount of data produced from the edge, the data transfer speed became a bottleneck. This degradation of speed affects the time in which the data is sent to the cloud, thus, causing the response time to be very long. On the other hand, edge computing provide faster processing as the source devices are already close to the edge, thus the transfer time is much less and consequently the overall response time will be less [28].
2. **Security and privacy**: For some applications, privacy is a main concern, by having the computation done at the edge privacy can be enforced prior to sending the data to the cloud [31]. Moreover, one of the disadvantages of cloud computing is that it is centralized which makes it a single point of failure and in case it is attacked, the security of all the devices communicating with it might be compromised.
3. **Scalability:** Another benefit of edge computing is that it provides scalability as the edge will be processing data from a small set of devices only. Moreover, when integrating edge computing with cloud computing the edge will take the responsibility of some of the processing that should be done and will send the

results only to the cloud which will reduce the bandwidth consumption and allow more devices to send their data [31].

2.3.Integrating Edge Computing and Blockchain

Integrating Blockchain and edge computing would merge the benefits of both of them; providing the privacy and security features of Blockchain as a distributed ledger and the ability of edge computing to carry out computing, storage and networking tasks at the network edge [30].

In this work, we integrated the Edge-based Electroencephalography (EEG) classification done by Awad et al. in [32] with our Blockchain framework to ensure QoS. The authors in [32] propose a smart monitoring health system for epileptic seizure detection at the network edge by classifying EEG signals as they proved to have a significant contribution in diagnosing several brain disorders [33]. The proposed system consists of three layers, i.e. IoT devices, edge, cloud. The edge layer, represented by a smartphone in their work, has three main tasks: acquiring the data from IoT devices, feature extraction and classification, and data reduction. To allow reliable detection, the authors apply Time-Domain (TD) and Frequency-Domain (FD) feature extraction and use these features to develop a fuzzy classification technique for EEG signals. They used the dataset in [34] considering three patients' classes. Each of these classes has 100 EEG segment and 4096 samples with a sampling rate of 173.6 sample/s and a sensing time frame of 23.6s:

1. Seizure-free (SF): Normal patients that have no seizures
2. Non-active (NAC): Patients that had a seizure, but it is not active
3. Active (AC): Patients with active seizure

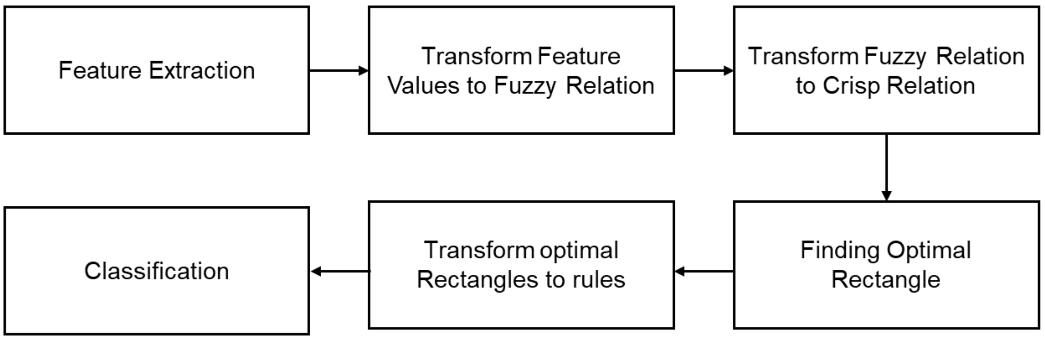
These three classes reflect different emergency levels and different channel requirement as shown in table 1.

Table 1: Emergeny Implication of Signal States

Class	Emergency	Channel Requirements
Active (AC)	Very High	Guaranteed service with minimal latency
Non-active (NAC)	Moderate	Some delay is acceptable
Seizure-free (SF)	Low	Transactions can handle higher latency

2.3.1. Classification Module

In this section, the developed rule extraction and classification done in [32] are explained. As shown in fig. 7, the rule extraction and classification process consist of 6 steps.

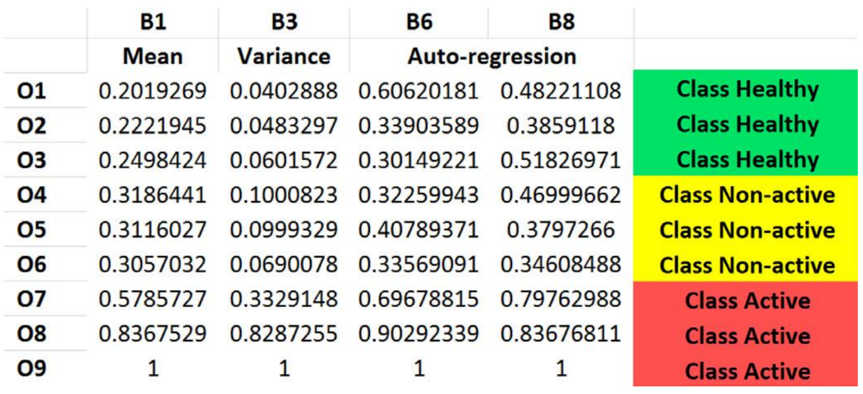

Feature Extraction
Transform Feature Values to Fuzzy Relation
Transform Fuzzy Relation to Crisp Relation
Finding Optimal Rectangle
Transform optimal Rectangles to rules
Classification
Figure 7: Rule extraction and classification Steps [32]

- **Step 1: Feature Extraction**

In this step a set of epileptic features are identified to differentiate between the EEG signals, these features are divided into two categories: Time-Domain (TD) features and Frequency-Domain (FD) features. Firstly, four TD features are considered which are: the absolute mean of the signal, the variance, the waveform length and finally the auto-regression coefficients. Secondly, for the FD feature extraction, the signals were initially converted to FD using Fast Fourier Transform (FFT) which revealed an important property which is the amplitude range as each of the classes appeared to have different amplitude range. The signals were divided into five frequency sub-bands: $\delta\ (0.2-3)Hz,\ \theta\ (3-8)Hz,\ \alpha\ (8-12)Hz,\ \beta\ (12-32)Hz\ and\ \gamma\ (>32)Hz$. Both TD and FD features were calculated for three patients from each class (i.e. SF, NAC and AC), then, only the features with high correlation with its class label and low correlation with the other classes are chosen, which are the mean, variance and the auto-regression features.

- **Step 2: Transform Feature Values to Fuzzy Relation**

Let $U = O \times P$ be the universe, where O is the set of all patients and P is the set of all features. The fuzzy set R on the universe U represents the strength of the relation by a membership value $\mu_R(o,p)$ between any pair (o,p) where o is the patient and p is the feature, $o \in O$ and $p \in P$. To convert the feature values to fuzzy relation, first, the absolute values of the features are taken. Then, the values are normalized in order to map the selected features to values within [0,1] range (fig. 8).

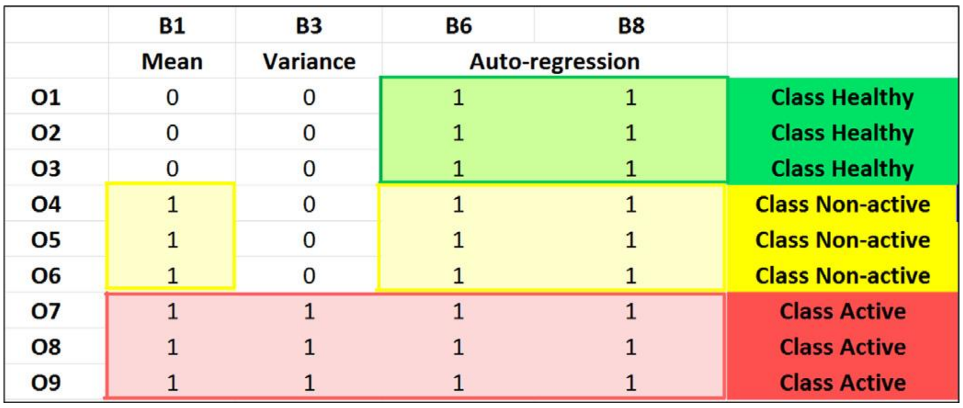


	B1	B3	B6	B8	
	Mean	Variance	Auto-regression		
O1	0.2019269	0.0402888	0.60620181	0.48221108	Class Healthy
O2	0.2221945	0.0483297	0.33903589	0.3859118	Class Healthy
O3	0.2498424	0.0601572	0.30149221	0.51826971	Class Healthy
O4	0.3186441	0.1000823	0.32259943	0.46999662	Class Non-active
O5	0.3116027	0.0999329	0.40789371	0.3797266	Class Non-active
O6	0.3057032	0.0690078	0.33569091	0.34608488	Class Non-active
O7	0.5785727	0.3329148	0.69678815	0.79762988	Class Active
O8	0.8367529	0.8287255	0.90292339	0.83676811	Class Active
O9	1	1	1	1	Class Active

Figure 8: Converting TD feature values to fuzzy relation [32]

- **Step 3: Transform Fuzzy Relations to Crisp Relation**

In this step, the obtained fuzzy relations R values are converted to crisp relation R_α, based on a certain $\alpha - cut$, $\alpha \in [0,1]$ such that for any $(o,p) \in U$, $\mu_{R\alpha}(o,p) = 1$, if $\mu_R(o,p) \geq \alpha$, otherwise, $\mu_{R\alpha}(o,p) = 0$. Fig. 9 shows the conversion of the fuzzy relation to crisp relation using $\alpha = 0.3$



	B1	B3	B6	B8	
	Mean	Variance	Auto-regression		
O1	0	0	1	1	Class Healthy
O2	0	0	1	1	Class Healthy
O3	0	0	1	1	Class Healthy
O4	1	0	1	1	Class Non-active
O5	1	0	1	1	Class Non-active
O6	1	0	1	1	Class Non-active
O7	1	1	1	1	Class Active
O8	1	1	1	1	Class Active
O9	1	1	1	1	Class Active

Figure 9: Converting TD fuzzy to crisp relation and finding optimal rectangle [32]

- **Step 4: Finding Optimal Rectangles**

Getting the binary crisp relation paves the way for identifying optimal rectangles that covers maximum number of patients that share the maximum number of features from the same class**.** As illustrated in fig. 9, there are three optimal rectangles, each corresponds to a class.

- **Step 5: Transform Optimal Rectangles to Rules**

In this step, the identified rectangles are used to set the association rules that will classify the data into the three classes in both TD and FD features. The conditions of the rules are decided based on the features within the rectangle and the classification decision reflects the corresponding class. Table 2 shows the association rules generated from the TD features.

Table 2: Association Rules Generated from TD Features [32]

Rule	Class
If $B_1 = 0$ AND $B_3 = 0$ AND ($B_6 = 1$ OR $B_8 = 1$)	SF
Else If $B_3 = 0$	NAC
Else If $B_1 = 1$ AND $B_3 = 1$ AND $B_6 = 1$ AND $B_8 = 1$	higher latency

- **Step 6: Classification**

Finally, the association rules are used in implementing the classifier on the smartphone to be applied on the data received from the patients' devices.

After the classification is done, the data will be sent to our multi-channel Blockchain depending on its priority (See Chapter 4).

3. Literature Review

In the last decade, the data generated by the healthcare sector increased remarkably [35]. This growth in the amount of data generated will continue to increase through the years specially in the era of IoT that witnesses a high demand for wearable devices operated in healthcare monitoring. It is expected that by next year 50.2 million patients will use remote health monitoring. As these numbers are increasing, one of the main concerns is efficient secure healthcare data sharing [36]. Permissioned blockchains act as a secure environment for storing data as it provides integrity, confidentiality and availability. This section explores some of the work done in the area of storing and sharing healthcare data using Blockchain.

For sharing Personal Health Information (PHI), [37] proposes Blockchain-based Secure and Privacy-Preserving PHI sharing (BSPP). The proposed scheme consists of two blockchains: 1- A private Blockchain within each hospital 2- A consortium Blockchain where all the hospitals participate in. The main aim of the private Blockchain is to store the PHI of each patient, on the other hand, the consortium is utilized to store only keywords that are used to search the patient's information that is stored in the private Blockchain. The designed system has three main entities:

1. **System Manager**: It takes the responsibility of registering the users (patients and doctors) by generating their public keys.
2. **Hospitals:** Each hospital has a server and a number of clients:

a. Server: It keeps the register table of the users. Moreover, it collects new blocks from the private blockchain and formulates new blocks for the consortium, it also works on verifying the consortium's blocks. Furthermore, the server authenticates the doctors outside the private blockchain to access the patient's PHI.

b. Clients: Used by the doctors to enter patient's data and create new blocks

3. **Patients:** Before meeting the doctor, patients must register to the server to get a token. This token is later used as a proof of the interaction between the patient and the doctor allowing the latter to create the patient's PHI.

Figs. 10a and 10b show the structure of the private Blockchain and consortium, respectively. For simplicity, the block headers, timestamp and issuer signature are eliminated. The private Blockchain blocks the issuing doctor's ID, patient's ID, his/her encrypted PHI and the keywords needed to search for it. The consortium's block stores the issuing server ID and secure indices, which consists of n transactions. Each of these transactions has the Block ID in which the PHI is stored, PHI keywords and patient's ID.

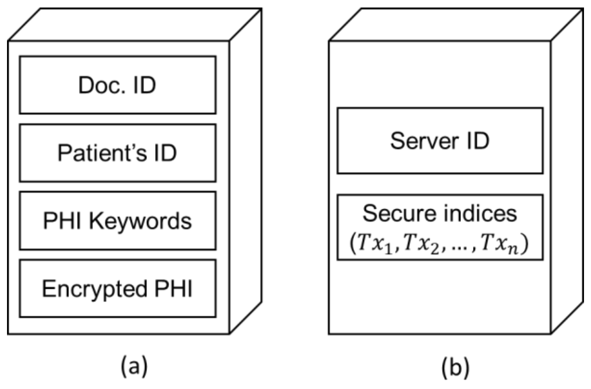


Figure 10: Block structure in a) Private blockchain b) Consortium blockchain [37]

In [38] the authors propose a Blockchain based storage scheme for healthcare data. The system has three main participants with different access permissions: hospitals, patients and third-party agencies. The main role of the hospitals is generating the medical records of the patients. Third-party agencies represent institutions responsible for appointment registration, hospital recommendations services, etc. The third participant, which are the patients, have control over their data as they can provide access to other participants to access their data as needed. Because of scalability and capacity issues, only indices are stored in the Blockchain, while the raw data is stored on cloud. The proposed system has three functionalities:

1. **Data Release**: Beside generating the data, the doctor also generates its hash digest then post it on the Blockchain after signing it with his/her private key. Additionally, an encrypted copy of the data is sent to the patient along with its encrypted encryption key. It is worth noting that this key is encrypted using the patient's public key.
2. **Data Storage**: To store the data, the patient verifies the signature of the issuing doctor. Then, the patient's private key is used to decrypt the encrypted encryption key to be able to decrypt the original data. Finally, the patient encrypts the data again with a new key and post it in the cloud.
3. **Data Sharing**: data sharing is under the control of the patient; access can be given to any institution by providing them with decryption key. The control policy defines the location, the access levels and the expiration data for accessing the data.

In [39], the authors propose a user-centric Blockchain-based system that collects medical data from the user's wearable devices, manual data entry and medical devices then send them to the cloud. The system incorporates six parties:

1. **Users:** Users are the owner of data in which it is collected from them through

their wearable devices, medical devices or through manual entry by their doctors. They are responsible for granting or denying access to other parties

2. **Wearable Devices:** They take the responsibility of collecting the data and transfer it to human readable format and then synchronized with their associated user account.
3. **Healthcare Providers:** Represents the doctors who are appointed by the users to do their medical tests and provide treatments. These doctors are given access under the user's permission.
4. **Health Insurance Companies**: The user can request a quotation from the insurance companies. To provide the quote, the insurance companies request data access from the user to check his/her medical health history and wearable devices data. In this case the user cannot deny access to the insurance companies to avoid fraud.
5. **Blockchain**: It used to store the data collected by the wearable devices, healthcare providers and insurance companies' quotation. In addition, all the access requests are stored in the blockchain.
6. **Cloud Database**: It is the point of communication between the users and the Blockchain. It is a client application that the user, healthcare provider or the insurance company interact with to update the ledger or read from after validating the access permissions.

CHAPTER 3: PERFORMANCE EVALUATION OF HLF

As explained in chapter 2, HLF is a distributed permissioned Blockchain that is highly customizable. In this chapter, HLF performance is studied by conducting several experiments to measure how the *batch-timeout*, the *batch size* and the *number of endorsers* would affect the *end-to-end latency* and the *throughput*.

1. Building the Network

In this section, the main components of building an HLF network are discussed. Building HLF network is divided into two main parts:

1- Network Infrastructure

2- Application Layer

1.1.Network Infrastructure

As shown in Fig. 11, healthnet network consists of one channel that connects two peer organizations: org1 and org2 and one orderer, each of the peer organizations has 2 connected peers. These peer organizations can be representing any institution such as hospital, pharmacy, etc. The ordering service utilizes the Solo-based design as it will not be used for production.

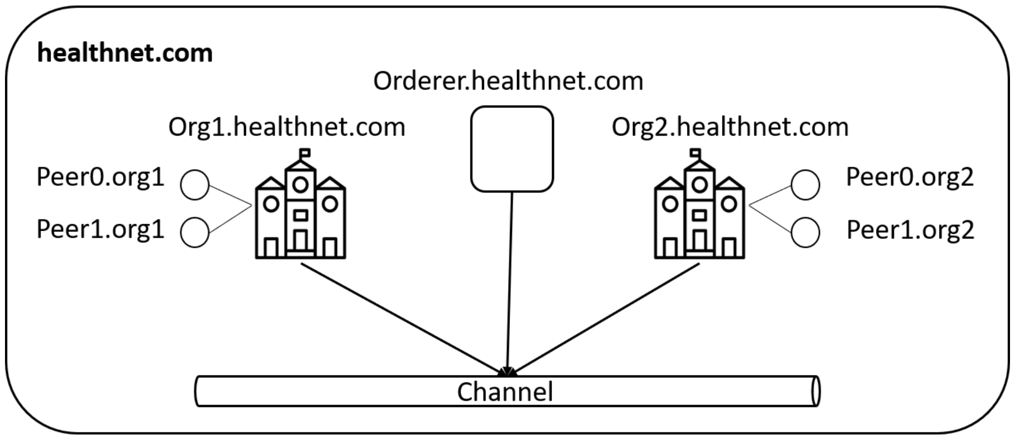


Figure 11: High level architecture of the network

The channel Configuration is defined by two main parameters, Batch-timeout and Batch Size:

- **Batch-timeout**: It defines the orderer's waiting time before creating block
- **Batch Size**: It controls the number of transactions per block; it is defined by three variables:
    1. Maximum transaction Count: The maximum number of transactions per block
    2. Absolute Maximum Bytes: The maximum number of bytes per block that cannot be exceeded
    3. Preferred maximum bytes: The preferred number of bytes per block

The aforementioned parameters have big impact on the performance of the network. The transactions are batched as a block whenever one of the limits is reached; meaning that if the batch-timeout is reached but the number of transactions is still less than batch size then, the orderer will batch the transactions into a block as discussed in chapter 2.

1.2.Application Layer

In this work, Hyperledger Composer is used to ease the process of implementing the application layer. It helps in modeling the business network that is being packaged to an archive (.bna file) to be used on top of HLF infrastructure. The business network is defined using three main files: Model File, Script File and Access Control File [40]:

- **Model File**: A .cto file that defines all the assets, participants and transactions. It is written in Hyperledger Composer Modeling Language.
- **Script File**: A .js file that is considered as the smart contract where the transactions logic is implemented, it is written in JavaScript.
- **Access Control File**: In HLF, users do not have the same access level. A .acl file defines the access control rules that states the CRUD (Create, Read, Update

and Delete) operations a user can perform in the business network like creating assets, participants or transactions. For example, some users can only read data, others can read, write create and delete.

Our application is designed for sharing medical data between healthcare participating entities. The model file defines one asset, two participants, and three transactions as shown in table 3.

Table 3: The Business Model

Assets	participants	Transactions
Case	Doctor AdminStaff	CreateDoctor CreateCase TransferCase

It is worth noting that in such scenarios, the business model should be more complicated, however, the main aim of this work is evaluating the network. Regarding the access control rules, only the Network Admins are allowed to add new assets, participants, and transactions. In real networks, the access rules have to be more complicated. For example, only the Network Admins can add new Staff, e.g., new doctors using the CreateDoctor transaction. Regarding the assets, only the doctors will be allowed to add a new Case using CreateCase transaction. Both Admins and doctors will be allowed to transfer a case from a doctor to another doctor. Each of the cases is defined by some attributes like the name of the patient, age, vitals, description, a supervising doctor, etc. It should be noted that all the experiments in this study were done using the createCase transaction to guarantee that all the transactions have the same size.

1.2.1. Transaction logic

- **CreateCase**:

1. Create a new Case instance
2. Assign the attributes to the case instance
3. Assign a doctor to the case
4. Add this case to the doctor's list
5. Update Case Registry
6. Update Doctor Registry

- **CreateDoctor**:

1. Create a new Doctor instance
2. Assign the attributes to the doctor instance
3. Update Doctor Registry

- **TransfereCase**:

1. Assign the case to another doctor
2. Add this case to the doctor's list
3. Update Case Registry
4. Update Doctor Registry

2. Experiments

Seven different experiments were conducted to show how HLF performance is affected by changing the batch-timeout, batch size (by changing the maximum transactions count) and the number of endorsing peers. Table 4 shows the exact setup, in experiments 1, 2 and 3 the batch size is fixed to 45 transactions (Tx) and the number of endorsers is fixed to 4 to examine the effect of changing the batch-timeout. Experiments 4, 5 and 6 study the effect of the batch size by fixing the batch-timeout to 200s and the endorsers to 4 and changing the batch size only. In experiments 1 and 7 the batch-timeout and batch size are set to 2s and 45 Tx, respectively, and only the number of endorsing peers changes. For each of the experiments the average end-to-end latency and the average throughput are measured by taking the average of 10 trials while having different number of parallel transactions. To study the effect of the batch-timeout and the number of endorsers, we started from 1 transaction up to 30 parallel transactions, however, while studying the effect of the batch size we started from 10 parallel transactions and not from 1 transaction; because if there is only one transaction then the batch size will never be reached and the delay will be caused by the batch-timeout which is set to 200s.

Table 4: Experimental Setup

Exp#	Batch-timeout	Batch Size*	Number of Endorsers
Exp. 1	2 seconds	45 Tx	4 endorsers
Exp. 2	5 seconds	45 Tx	4 endorsers
Exp. 3	8 seconds	45 Tx	4 endorsers
Exp.4	200 seconds	2 Tx	4 endorsers
Exp.5	200 seconds	5 Tx	4 endorsers
Exp. 6	200 seconds	10 Tx	4 endorsers
Exp. 7	2 seconds	45 Tx	2 endorsers

*Batch size was changed by changing the maximum number of transactions per block

The end-to-end latency and throughput are defined as follows:

- **End-to-end latency (s)**: The total time needed by the transaction to be committed to the ledger, it starts once the transaction is sent until it is committed to the ledger.
- **Throughput (transaction/s)**: Number of transactions that can be processed in one second, it is measured using the following equation:

$$\frac{No. of\ parallel\ transactions}{Latency}$$

As shown in table 5, in this phase, Fabric v1.2.0 was used on Ubuntu 16.04 LTS platform with processor Intel Core i7-4510U.

Table 5: System Specifications – Phase 1

Specification	Details
Operating System	Ubuntu 16.04 LTS
Processor	Intel Core i7-4510U CPU @ 2.00 GHz x 4
Fabric Version	1.2.0
Application Layer	Hyperledger Composer

3. Results

The experiments done are divided into three sets, the first set shows the effect of changing the batch-timeout, the second set studies how the block size affects the performance and the third set focuses on the number of endorsers.

3.1.Batch-timeout

In experiments 1, 2 and 3 we study how the end-to-end latency and throughput are affected by changing the batch-timeout as the number of concurrent transactions increases to 30 while fixing the batch size to 45 Tx which is big enough to avoid reaching the maximum batch size before the timeout and to accommodate all the transactions in one block. Fig. 12 shows that the latency increases as the number of parallel transactions increases. It also shows that it increases by increasing the batch-timeout. Fig. 13 shows that the throughput increases as the number of parallel transactions increases; this happens because the number of transactions within the block increases meaning that more transactions will be processed at a time. It is also observed that as the batch-timeout increases the throughput decreases as a result of increasing the delay.

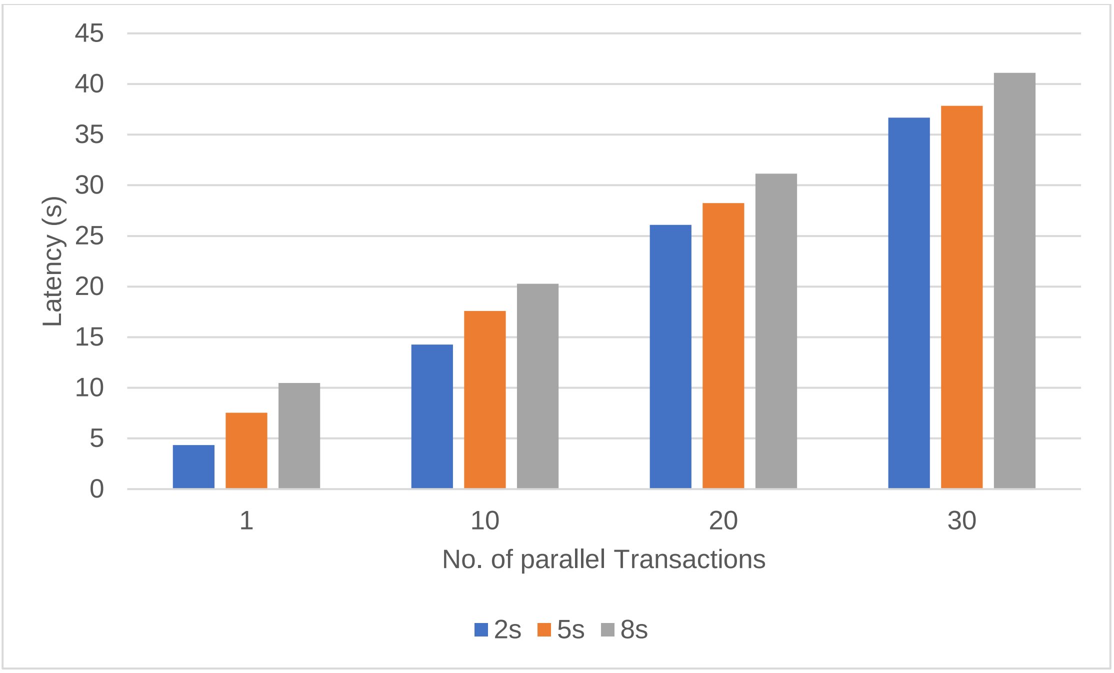


Figure 12: Average latency for varying batch-timeout

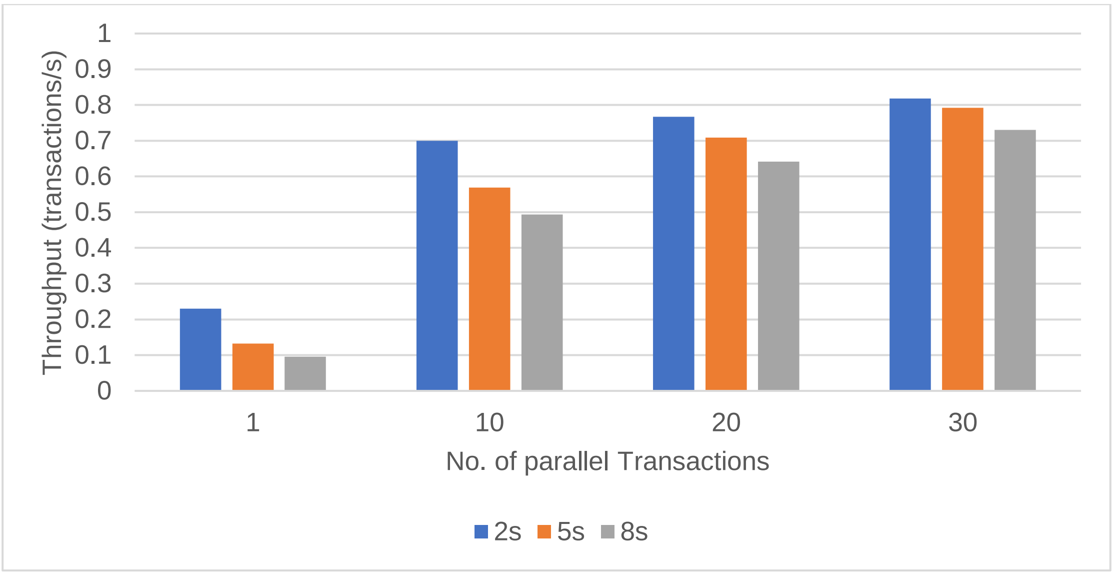


Figure 13: Average throughput for varying batch-timeout

3.2.Batch Size (Max. Number of Transactions per Block)

In experiments 4, 5 and 6 the batch-timeout is fixed to 200 seconds and the batch size varies by changing the maximum number of transactions per block to study how it affects the end-to-end latency and throughput. The batch-timeout was set to a high value

(200 seconds) to ensure that the blocks will never timeout before reaching the block size. Fig. 14 shows that the latency almost doubles when the number of transactions increases. It also shows that increasing the batch size decreases the latency; this change is clear when the batch size increases from 2 Tx to 5 Tx, but as it increases from 5 Tx to 10 Tx the change is negligible. The reason behind that is illustrated in fig. 15 which shows that the number of blocks in which the transactions were packaged and committed; the difference between the number of blocks committed is small when the batch size increased from 5 Tx to 10 Tx when it is compared to the difference between the number of blocks when it increased from 2 Tx to 5 Tx. This is reflected on the latency as it increases as the number of blocks increases because there are more blocks to be validated as discussed in chapter 2. Fig. 16 shows that the throughput decreases as the number of parallel transactions increases. On the other hand, it demonstrates that the throughput increases as the batch size increases.

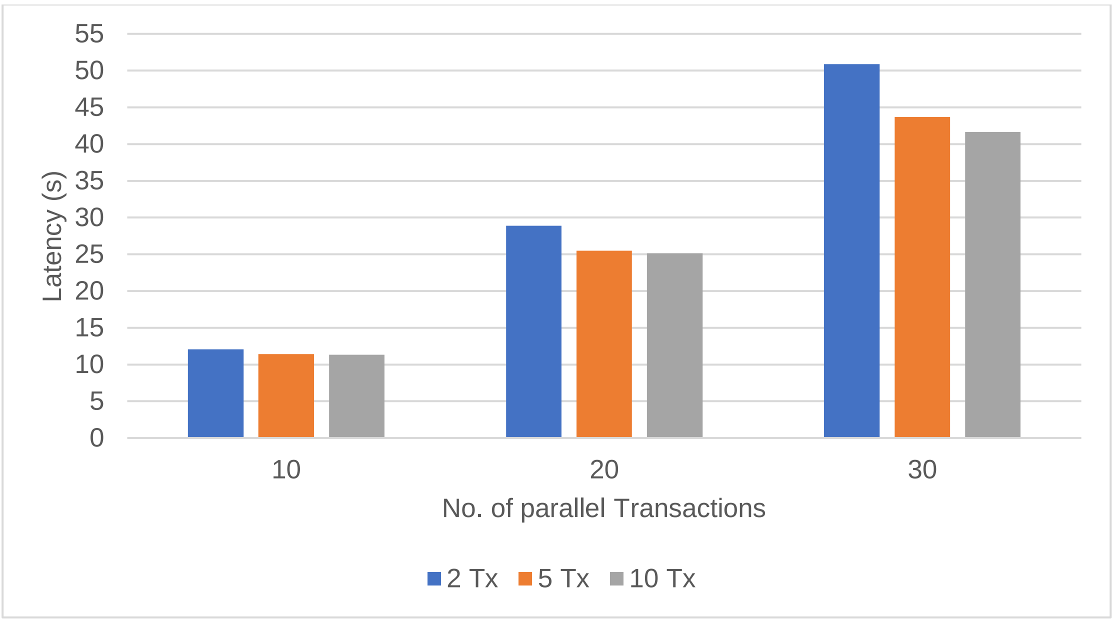


Figure 14: Average latency for varying batch Size

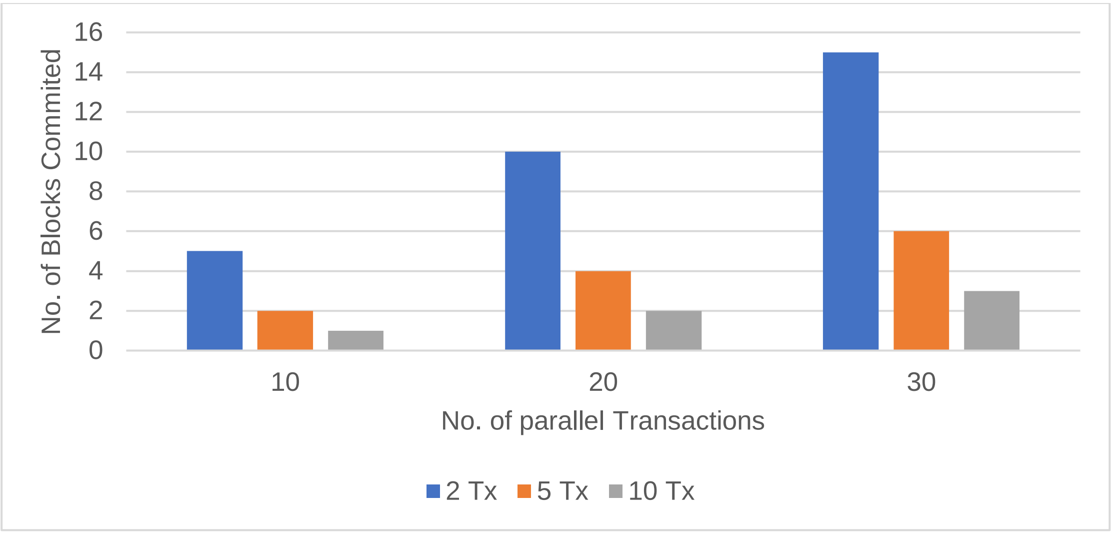


Figure 15: Number of blocks commited

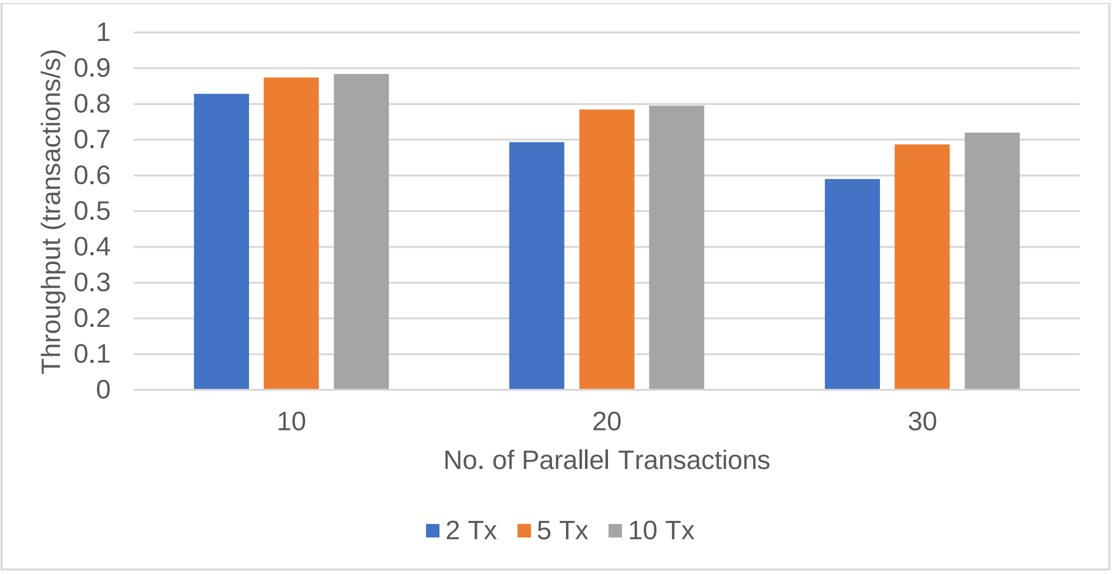


Figure 16: Average throughput with varying batch size

3.3.Endorsing peers

In experiments 1 and 7 we examine how the number of endorsing peers affects the latency. In both experiments, the batch-timeout is set to 200 seconds and the batch size is set to 45 Tx. From Fig. 17, it is observed that the latency increases as the number of parallel transactions increases. It also shows that increasing the numbers of endorsers leads to a slight increase in the latency, because with increasing the number of endorsers the client has to wait for more endorsed transactions responses before sending to the orderer in the second phase of the transaction flow.

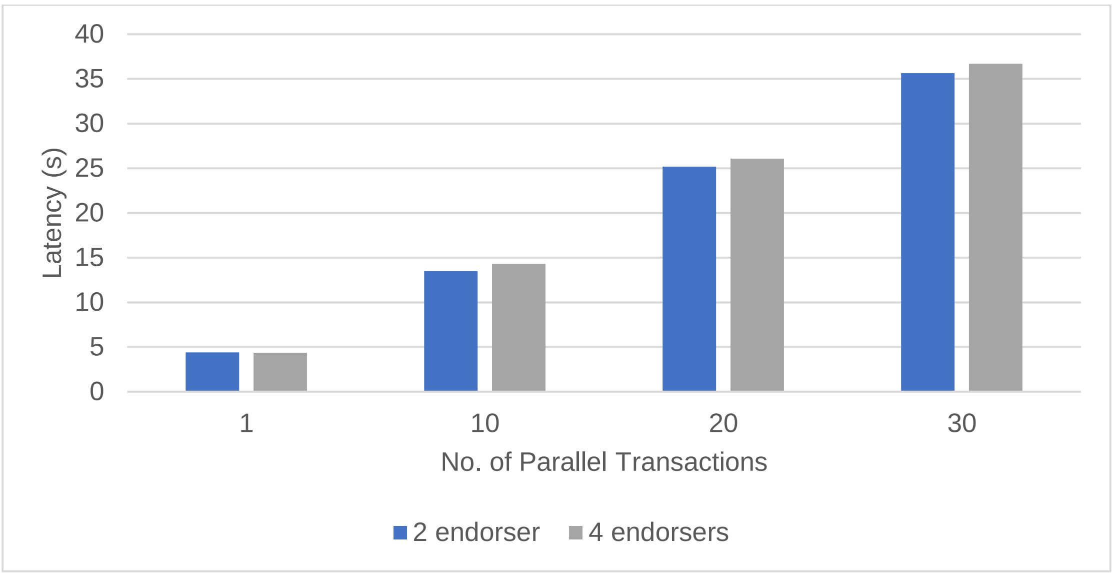


Figure 17: Average latency for varying number of endorsers

From the results presented, we deduce that applications with large number of parallel transactions, the batch-timeout and block size should be large in order to maintain high throughput while for application with urgent transactions, the batch-timeout and block size should be limited in order to obtain low latency (especially, in case of emergency).

CHAPTER 4: ENSURING QOS ON BLOCKCHAIN

Based on the analysis done in chapter 3 that shows that different channel configurations affect the performance of HLF blockchain in terms of end-to-end latency, we propose merging edge computing with HLF framework to ensure Quality of Service (QoS). In this chapter, a multi-channel Blockchain framework is built and integrated with edge computing, where the gateway formulates the transaction and decides on which channel it will be sent based on its priority.

1. System Architecture

As shown in fig. 18, the proposed system is divided into two parts:

1- A multi-channel Blockchain framework

2- Edge computing for classifying the transactions

The edge gateway collects the data either from the connected devices or from data entered manually by the users at each institution, then, it formulates the transaction and sends it to the suitable blockchain channel based on its priority. The proposed blockchain framework consists of three different channels each has its own configuration implying different behavior.

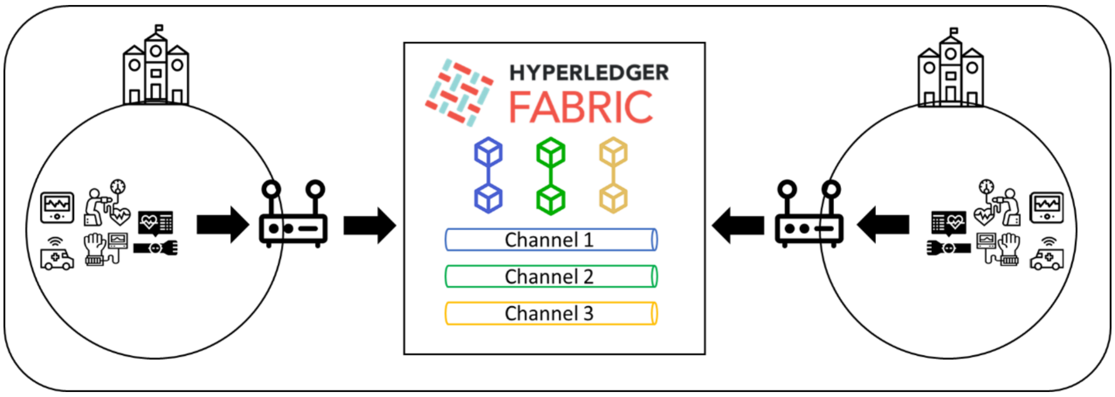


Figure 18: System architecture

1.1.Multi-channel Blockchain framework

1.1.1. Network Infrastructure

As shown in fig. 19, the network consists of three channels connecting two peer organizations: org1 and org2 and a Raft-based ordering service with 5 orderers. Each channel has different configuration, based on the urgency of the transaction the edge gateway will decide the channel in which the transaction will be sent to. Having three channels implies having three ledgers, and since all the peers are connected to the three channels, each peer will have a copy of the 3 different ledgers.

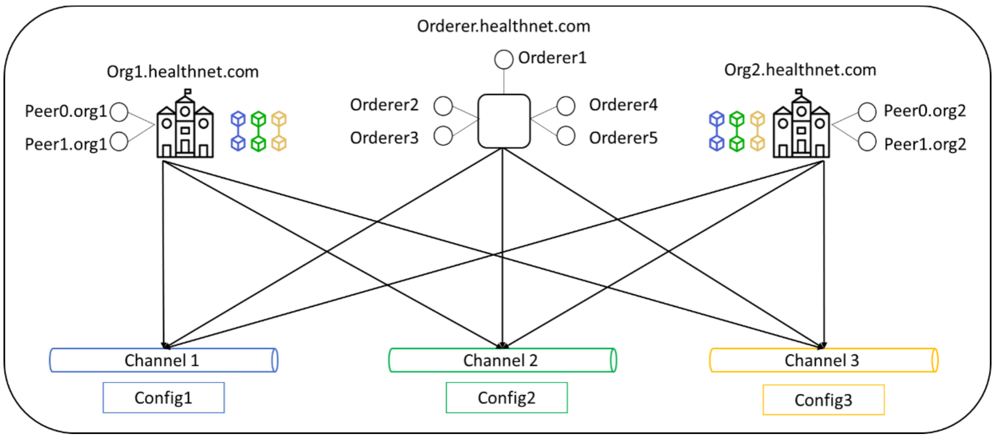


Figure 19: Multi-channel blockchain framework

Table 6 matches the channels to the emergency level and shows how the gateway will react to different emergency levels. If the emergency of the transaction is high, it will be sent to channel 1, if it is low it will be sent to channel 3 as it can tolerate some latency and if it is moderate then it will be sent to channel 2.

Table 6: Matching Channels to Transactions' Emergency

Channel No.	Emergency
Channel 1	Very High
Channel 2	Moderate
Channel 3	Low

Based on the performance evaluation done in chapter 3, we enforced different channels behavior by varying the batch-timeout and the block size (by changing maximum number of transactions per block). The results in chapter 3 showed that transactions that require low latency should have small batch-timeout and small block size while transactions that can tolerate delay should have higher batch-timeout and bigger block size. Accordingly, we set minimal batch-timeout to channel 1 as it requires very low latency while channels 2 and 3 have higher batch-timeout and block sizes as they can tolerate the delay.

To implement multiple channels, initially, the three channels are given the same configuration, then the channels are updated using a configuration block, the detailed implementation steps are explained in Appendix A.

1.1.2. Application layer

For the application layer, we built the chaincode natively on HLF infrastructure without using hyperledger composer as an intermediary layer as it does not support multiple channels.

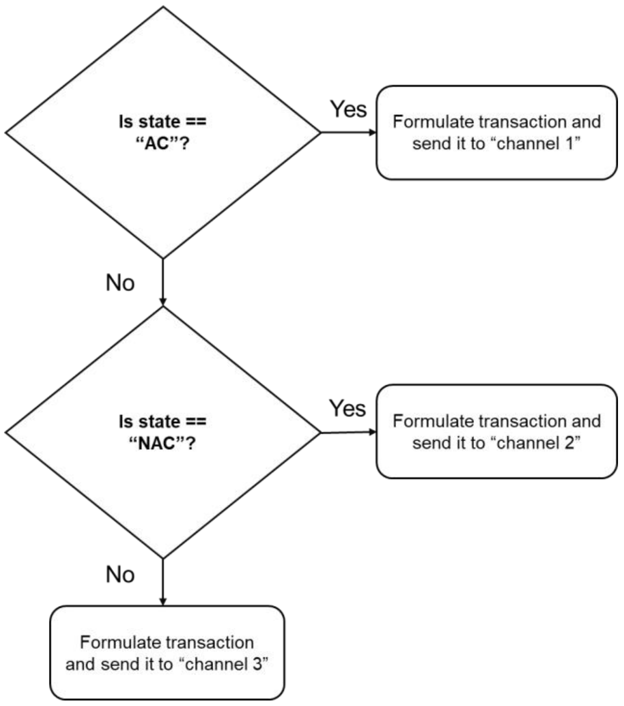
For developing our patient chaincode, Golang programming language is used. It mainly has one transaction which is createPatient transaction, similar to the createCase transaction in chapter 3. Also, the patient class has the same attributes that the Case Asset has such as: personal information, vitals, description, etc. but additionally, it also has EEG and state attributes.

After developing the chaincode, first, it has to be installed on the peers, this can be thought of as being physically hosted on the peer. Then, it is instantiated on the channel meaning making it logically available on the channel and accessible by its members, these steps are covered in more details in Appendix A.

1.2.Edge Computing

The main aim of this phase is classifying the transactions and mapping them to a specific class of service (i.e. channel) based on their emergency. This provides QoS and ensures that transactions with high priority have guaranteed service with minimum latency, while less important transactions can handle latency.

Fig. 20 shows how the edge gateway sends the data to the Blockchain. In this work we integrated the EEG classification done in [32] with our multi-channel Blockchain (See chapter 2, section 2.3.1). Based on the classification done, if the data is classified as AC, then it has high priority and it will be sent to channel 1 which provides low latency. If the data is classified as NAC, then it will be sent to channel 2, otherwise it will be sent to channel 3.



Is state == "AC"?
Yes
Formulate transaction and send it to "channel 1"
No
Is state == "NAC"?
Yes
Formulate transaction and send it to "channel 2"
No
Formulate transaction and send it to "channel 3"

Figure 20: Deciding the transmission channel at the edge gateway

2. Experiments

In this chapter, three experiments are considered; the first two experiment are applied on the three channels and the third experiment is applied on the second and third channels only. Table 7 shows the configurations of each channel, channel 1 should be used for urgent transactions; thus, the batch-timeout is set to 2 seconds and the block size (by setting the maximum number of transactions) is set to 10 transactions. For channel 2, the batch-timeout is set to 200 seconds as it can tolerate some latency and the block size is set to 10, which ensures that the block will be sent to the next step in case the batch reached 10 transactions even if the batch-timeout is not reached yet. Finally, channel 3 is responsible for the transactions that can handle high delay; thus, the batch-timeout is set to 200 seconds and the block size is set to 25 transactions. It is worth mentioning that in our study we considered only the batch-timeout and the maximum number of transactions per block while the absolute maximum bytes were fixed to the default value in all the channels.

Table 7: Channels Configuration

Parameter	Channel 1	Channel 2	Channel 3
Batch-timeout	2 Seconds	200 Seconds	200 Seconds
Maximum Transactions count	10 Tx	10 Tx	25 Tx
Absolute maximum bytes	99 MB	99 MB	99 MB

In the first two experiments 30 consecutive transactions are sent to the blockchain and the latency is calculated per transaction from the time it was submitted until the block is committed to the ledger. In the first experiment the time gap between each two transactions is set to 20 seconds, while in the second experiment the time gap starts with three seconds and keeps incrementing by 3 seconds until it reaches 84 seconds between the last two transactions. (3s, 6s, 9s, …84s). In the third experiment, we study how the latency in channels 2 and 3 is affected when the frequency of sending transactions varies. This is done by setting four different time gaps (5s, 10s, 45s and 50s), then measuring the average latency on both channels. In this experiment, the latency was measured relative to the time gaps, where the average latency for 30-time gaps was considered (i.e. 31 transactions).

As shown in table 8, in this phase, Fabric v1.4.4 was used on Ubuntu 16.04 LTS platform with processor Intel Core i7-4600U.

Table 8: System Specifications – Phase 2

Specification	Details
Operating System	Ubuntu 16.04 LTS
Processor	Intel Core i7-4600U CPU @ 2.10 GHz x 4
Fabric Version	1.4.4
Application Layer	Native Hyperledger Fabric

3. Results and Discussion

This section discusses the results of the three sets of experiments, the first experiment shows how the transaction latency is affected on each channel when the time gap between consecutive transactions is fixed. In the second set we study how the latency is affected on each of the three channels as the time gap between consecutive transactions varies. Finally, the third experiment shows how the latency is affected as the time gap between consecutive transactions increases which implies a decrease in the frequency of sending the transactions.

3.1.Fixed Time Gap

In this experiment a total of 30 transactions is sent to each channel in which a transaction is sent every 20 seconds. Fig. 21 illustrates that channel 1 has very low latency compared to channel 2 and 3, also, the latency is almost stable in this channel and the variance is very small as shown in table 9. This stability resulted from the small batch-timeout that guarantees that the orderer will send the block to the validation step withing a very short time period (i.e. 2 seconds). In channel 2 the latency keeps fluctuating; it starts with the maximum value and it keeps decreasing until it reaches its minimum value at the 10th transaction, this trend keeps repeating every 10 transactions. As the figure depicts, channel 2 never reaches the batch-timeout; meaning that the block is sent to the next step as a result of reaching the maximum number of transactions. Similar to channel 2, channel 3 has the same trend as it starts declining from transaction 1 until it reaches the 10th transaction then it peaks at 200 seconds again at the 11th transaction. Although channel 2 and channel 3 have the same trend, in channel 3 the block is sent to its next step because the batch-timeout is reached, while in channel 2, the block is sent to the next step because it reached its maximum block size before the batch-timeout is reached. This difference results in higher average in channel 3 as shown in table 9. The table also shows that the variance in channels 2 and 3 is almost the same however in channel 1 the variance is much smaller which ensures QoS and guarantees that urgent transactions will be committed within a short time period.

Table 9: Experiments 1 results (Fixed time Gap)

	Channel 1	Channel 2	Channel 3
Average	2.122367	91.95853	108.4802
Variance	0.000359	3542.672	3541.612
Standard Deviation	0.018944	59.52035	59.51144

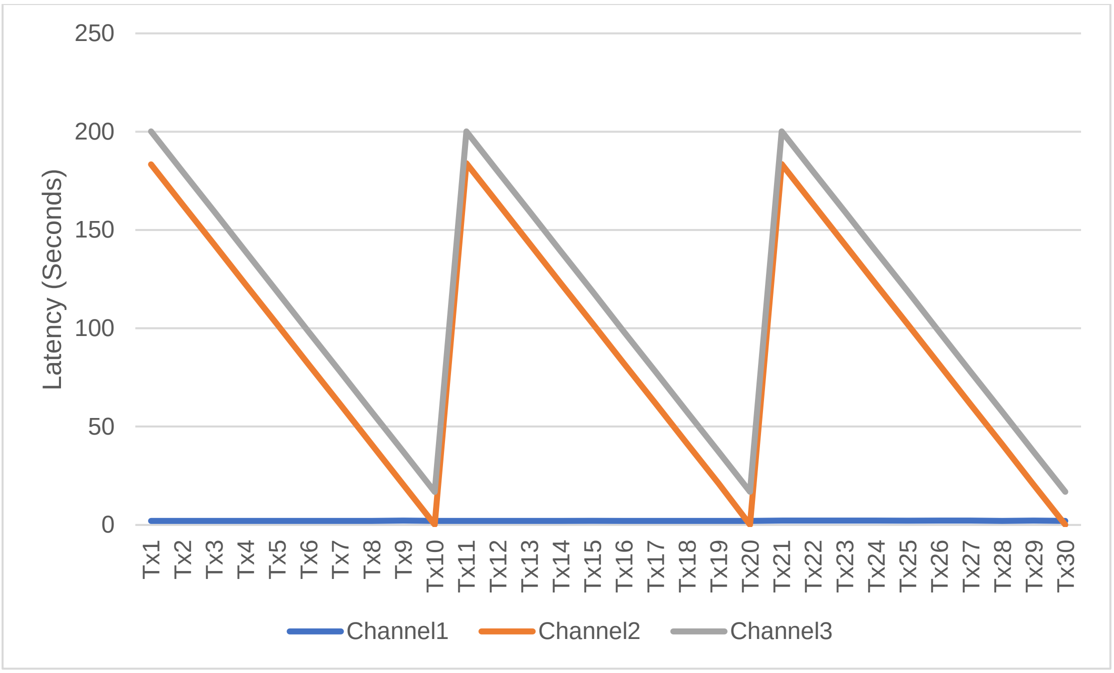


Figure 21: Transactions latency with fixed time gap

3.2.Varying Time Gap

Similar to the first experiment, 30 transactions are sent to the three channels, however, in this experiment the time gap between consecutive transactions increases by 3 seconds; meaning that the frequency of sending the transactions keeps decreasing. The results in fig. 22 shows that the latency in channel 1 is stable and the variance is very small as shown in table 10. In channel 2 and channel 3, the latency follows the same trend except that it has higher value on channel 3 when the frequency of sending transactions was high but as the gap between sending the transactions increases, the delay in both channels becomes almost the same. This change in the behavior happened because in the beginning, channel 2 was depending on the block size to start the next phase in the execute-order-validate process as it had the chance to reach the maximum block size before reaching the batch-timeout. On the other hand, channel 3 was depending on the batch-timeout because the maximum number of transactions per block is high, hence, the batch-timeout was reached first. By the time the frequency of sending transactions decreased, both channels started depending on the batch-timeout because the maximum block size in both channels was never reached within the given batch-timeout (i.e. 200 seconds) which resulted in the similar behavior.

Table 10: Experiments 2 Results (Varying time Gap)

	Channel 1	Channel 2	Channel 3
Average	2.119633	103.9158	124.3701
Variance	0.000246	4023.488	4066.6
Standard Deviation	0.01569	63.43097	63.7699

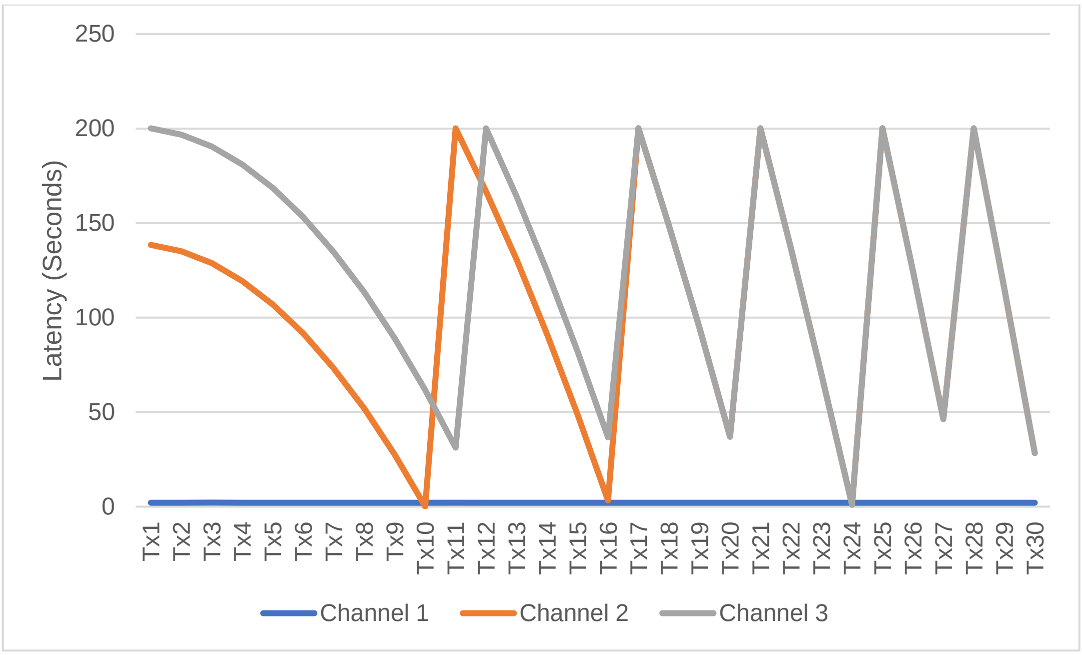


Figure 22: Transactions latency with varying time gap

3.3.Varying the Transactions Frequency

In this experiment, we study how the latency is affected as the number of transactions sent per time slot changes, this is done by changing the time gap between each two consecutive transactions. Four different values are considered in this experiment: 5 seconds, 10 seconds, 45 seconds and finally 50 seconds. As shown in fig. 23, the latency generally increases as the time gap increases. By considering the two channels, when the time gap is small (i.e. 5s and 10s), there is a considerable delay difference. However, when the frequency of sending transaction decreases as in the case of 45 s and 50s, there is almost no difference. In the case of short time gap, the block is batched based on the block size, thus, in channel 2, the orderer waits until it receives only 10 transactions while in channel 3 it should wait until it receive 25 transactions as defined in the channels' configuration. Although, it was expected that channel 3 will have its transactions batched when they reach 25 transactions, we noticed that it was affected by another factor which is the absolute maximum bytes resulting in three blocks (fig.24), where the first and second blocks have 15 transactions when the time gap was 5 seconds. What is more interesting is channel's 3 behavior when the time gap was 10 seconds, the first block was batched after 15 transactions similar to the first case, but the second block was batched after 5 transactions only, this happened because the first block did not reach either the maximum number of transactions nor the batch-timeout and it was batched because it reached the absolute maximum bytes. As a result, the second block did not reset its batch-timeout, and it batched the transactions when the batch-timeout of the first block was reached resulting in a block with a smaller number of transactions. Fig 24 also shows that as the frequency of sending the transactions decreases (i.e. longer time gap), the number of blocks created increases because the orderers start depending on the batch-timeout.

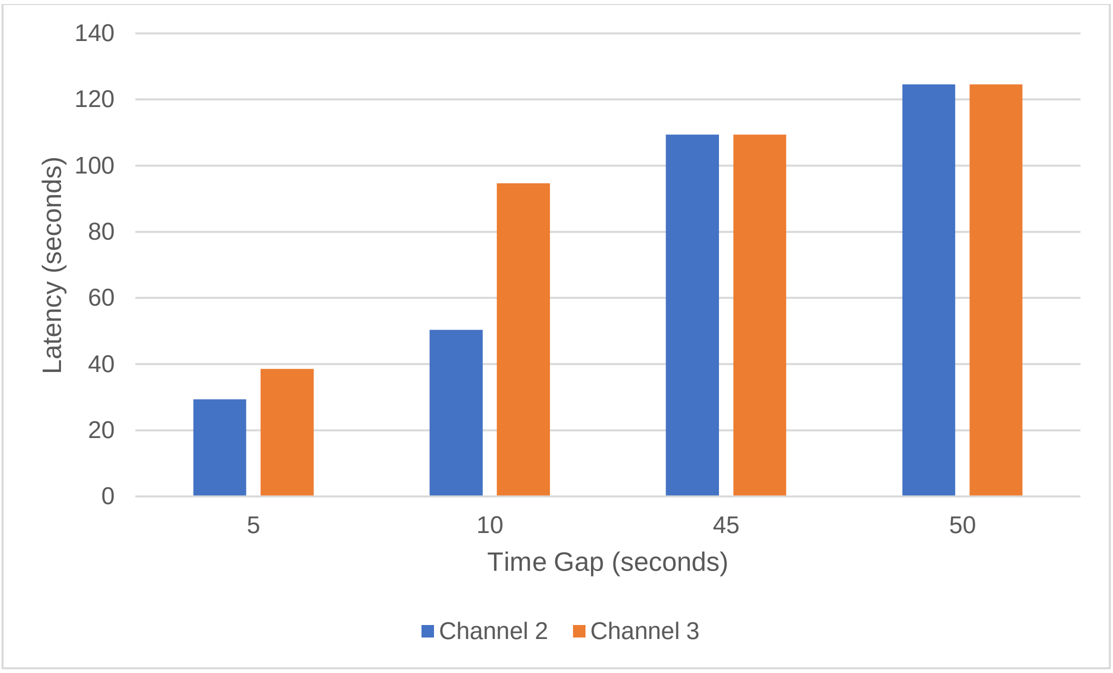


Figure 23: Effect of varying the transactions frequency

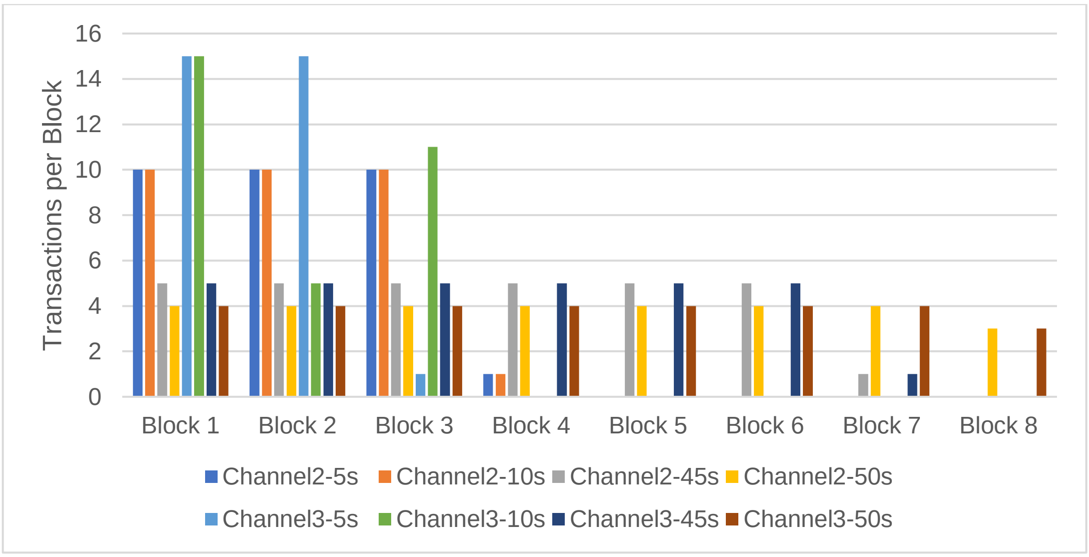


Figure 24: Number of transactions per block

The results of the conducted experiments prove that using multi-channel blockchain framework can provide QoS and give higher priority to some transactions over others. In our system, the gateway sends transactions with high priority, AC seizures in our case, to channel 1 which guarantees that the transaction will be committed to the ledger with minimum delay. In case the transaction is less urgent like NAC seizures it will be sent to channel 2 that has higher latency but the block size is not that high, hence, if the number of transactions sent exceeds 10, each 10 transactions will be processed to the validation phase once they reach the orderer. In the third channel, both the batch-timeout and the block size are high resulting in high latency, this could be suitable for SF patients.

CHAPTER 5: CONCLUSION AND FUTURE WORK

1. Conclusion

In this work we started by investigating the performance of HLF, where we conducted several scenarios to study the average end-to-end latency and throughput. The parameters that were considered in this phase are the batch-timeout, block size and the number of endorsing peers, while varying the number of parallel transactions. The results reveal that the latency increases as the number of transactions and batch-timeout increase. Also, we show that the number of generated blocks and number of transactions per block have an impact on the obtained throughput. Indeed, the throughput increases as the block size increases, because more transactions in one block means that more transactions will be validated at the same time. It is also observed that increasing the batch-timeout leads to an increase in the latency because each block has to wait for the timeout even if it has received all the transactions.

Based on these findings, we developed a multi-channel Blockchain framework and integrated it with edge based smart-health classification scheme that classifies the EEG signals into AC, NAC and SF patients then, maps the transactions to one of the channels based on their emergency level, thus ensuring QoS. Channel 1 has small batch-timeout and small block size guaranteeing that the transaction will be committed in a very short time regardless of the frequency of sending transactions. In the second channel, the batch-timeout is set to a high value while the block size is small, thus, if the number of transactions is a multiple of the maximum number of transactions per block, the transactions will be batched without waiting for the batch-timeout. The last channel has high batch-timeout and large block size, which makes the latency higher in this case.

The developed multi-channel framework proved to provide QoS and give higher priority to some transactions over others. The system gives higher priority to AC seizure as the patients in this case are in high risk which requires a quick action from the healthcare providers (i.e. Medical staff). For NAC patients, the transactions have less priority and can handle more latency, but they need to be sent periodically to the healthcare providers to monitor the patients. Finally, the SF patients has the lowest priority as they are in stable state.

2. Future Work

For the future work, more parameters will be studied, such as the effect of the number of orderers, and the number of validators, particularly on the security aspect of the transaction management process. Furthermore, the studies here can be extended to analyze the scalability issues by studying large size network and with increasing the number of organizations and parallel transactions. In such large-scale environments, the configurations for the individual channels can be estimated dynamically to optimize transaction QoS through addressing the trade-off between transaction latency and security. Security can be guaranteed through more rigorous transaction validation, which in turn affect the latency end-to-end. Therefore, addressing this trade-off through efficient algorithm and trying this on the experimental testbed, can be a topic of future work.

REFERENCES

[1] B. A. Tama, B. J. Kweka, Y. Park, and K.-H. Rhee, “A critical review of blockchain and its current applications,” in *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*, Aug. 2017, pp. 109–113, doi: 10.1109/ICECOS.2017.8167115.

[2] A. Goranović, M. Meisel, L. Fotiadis, S. Wilker, A. Treytl, and T. Sauter, “Blockchain applications in microgrids an overview of current projects and concepts,” in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2017, pp. 6153–6158, doi: 10.1109/IECON.2017.8217069.

[3] F. Dai, Y. Shi, N. Meng, L. Wei, and Z. Ye, “From Bitcoin to cybersecurity: A comparative study of blockchain application and security issues,” in *2017 4th International Conference on Systems and Informatics (ICSAI)*, Nov. 2017, pp. 975–979, doi: 10.1109/ICSAI.2017.8248427.

[4] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System” [Online]. Available: https://bitcoin.org/bitcoin.pdf. (accessed Nov. 05, 2019)

[5] V. Singla, I. K. Malav, J. Kaur, and S. Kalra, “Develop Leave Application using Blockchain Smart Contract,” in *2019 11th International Conference on Communication Systems Networks (COMSNETS)*, Jan. 2019, pp. 547–549, doi: 10.1109/COMSNETS.2019.8711422.

[6] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, “PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain,” p. 11.

[7] X. Xu., I. Weber, M. Staples, L. Zhu, J. Bosch, C. Pautasso and P. Rimba “A Taxonomy of Blockchain-Based Systems for Architecture Design,” in *2017 IEEE*

International Conference on Software Architecture (ICSA), Apr. 2017, pp. 243–252, doi: 10.1109/ICSA.2017.33.

[8] X. Zhang, R. Qin, Y. Yuan, and F.-Y. Wang, "An Analysis of Blockchain-based Bitcoin Mining Difficulty: Techniques and Principles," in *2018 Chinese Automation Congress (CAC)*, Nov. 2018, pp. 1184–1189, doi: 10.1109/CAC.2018.8623140.

[9] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[10] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake", self-published paper, 2012.

[11] "Peercoin University." https://university.peercoin.net/ (accessed Nov. 12, 2019).

[12] M. E. Peck, "Blockchain world - Do you need a blockchain? This chart will tell you if the technology can solve your problem," *IEEE Spectrum*, vol. 54, no. 10, pp. 38–60, Oct. 2017, doi: 10.1109/MSPEC.2017.8048838.

[13] "Home | Ethereum," *ethereum.org*. https://ethereum.org (accessed Nov. 13, 2019).

[14] "Corda | Open Source Blockchain Platform for Business," *Corda*. https://www.corda.net/ (accessed Mar. 13, 2020).

[15] "MultiChain | Open source blockchain platform." https://www.multichain.com/ (accessed Mar. 13, 2020).

[16] "Hyperledger Fabric," *Hyperledger*. https://www.hyperledger.org/projects/fabric (accessed Nov. 09, 2019).

[17] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance Analysis of Private Blockchain Platforms in Varying Workloads," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*,

Jul. 2017, pp. 1–6, doi: 10.1109/ICCCN.2017.8038517.

[18] “mining - ETHWIKI.” https://eth.wiki/mining/ (accessed Mar. 13, 2020).

[19] “ethereum/wiki,” *GitHub*. https://github.com/ethereum/wiki (accessed Mar. 13, 2020).

[20] S. Rouhani and R. Deters, “Performance analysis of ethereum transactions in private blockchain,” in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Nov. 2017, pp. 70–74, doi: 10.1109/ICSESS.2017.8342866.

[21] “Wood - ETHEREUM A SECURE DECENTRALISED GENERALISED TRANS.pdf.” Accessed: Apr. 05, 2020. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf.

[22] R. G. Brown, “The Corda Platform: An Introduction”, (accessed Apr. 05, 2020).

[23] “MultiChain-White-Paper.pdf.” Accessed: Mar. 13, 2020. [Online]. Available: https://www.multichain.com/download/MultiChain-White-Paper.pdf.

[24] E. Androulaki *et al.*, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” presented at the Proceedings of the Thirteenth EuroSys Conference, Apr. 2018, p. 30, doi: 10.1145/3190508.3190538.

[25] “hyperledger-fabricdocs Documentation,” p. 507.

[26] “Apache ZooKeeper.” https://zookeeper.apache.org/ (accessed Nov. 09, 2019).

[27] C. Martín Fernández, M. Díaz Rodríguez, and B. Rubio Muñoz, “An Edge Computing Architecture in the Internet of Things,” in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, May 2018, pp. 99–102, doi: 10.1109/ISORC.2018.00021.

[28] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct.

2016, doi: 10.1109/JIOT.2016.2579198.

[29] A. Nawaz, T. N. Gia, J. P. Queralta, and T. Westerlund, “Edge AI and Blockchain for Privacy-Critical and Data-Sensitive Applications,” in *2019 Twelfth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*, Nov. 2019, pp. 1–2, doi: 10.23919/ICMU48249.2019.9006635.

[30] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, “Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1508–1532, Secondquarter 2019, doi: 10.1109/COMST.2019.2894727.

[31] M. Satyanarayanan, “The Emergence of Edge Computing,” *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.

[32] A. Awad Abdellatif, A. Emam, C.-F. Chiasserini, A. Mohamed, A. Jaoua, and R. Ward, “Edge-based compression and classification for smart healthcare systems: Concept, implementation and evaluation,” *Expert Systems with Applications*, vol. 117, pp. 1–14, Mar. 2019, doi: 10.1016/j.eswa.2018.09.019.

[33] H. Adeli, S. Ghosh-Dastidar, and N. Dadmehr, “A Wavelet-Chaos Methodology for Analysis of EEGs and EEG Subbands to Detect Seizure and Epilepsy,” *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 2, pp. 205–211, Feb. 2007, doi: 10.1109/TBME.2006.886855.

[34] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, “Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state,” *Phys. Rev. E*, vol. 64, no. 6, p. 061907, Nov. 2001, doi: 10.1103/PhysRevE.64.061907.

[35] H. Kupwade Patil and R. Seshadri, “Big Data Security and Privacy Issues in

Healthcare," in *2014 IEEE International Congress on Big Data*, Jun. 2014, pp. 762–765, doi: 10.1109/BigData.Congress.2014.112.

[36] M. H. Kassab, J. DeFranco, T. Malas, P. Laplante, G. Destefanis, and V. V. Graciano Neto, "Exploring Research in Blockchain for Healthcare and a Roadmap for the Future," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019, doi: 10.1109/TETC.2019.2936881.

[37] A. Zhang and X. Lin, "Towards Secure and Privacy-Preserving Data Sharing in e-Health Systems via Consortium Blockchain," *J Med Syst*, vol. 42, no. 8, p. 140, Jun. 2018, doi: 10.1007/s10916-018-0995-5.

[38] Y. Chen, S. Ding, Z. Xu, H. Zheng, and S. Yang, "Blockchain-Based Medical Records Secure Storage and Medical Service Framework," *J Med Syst*, vol. 43, no. 1, p. 5, Jan. 2019, doi: 10.1007/s10916-018-1121-4.

[39] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, "Integrating blockchain for data sharing and collaboration in mobile healthcare applications," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Oct. 2017, pp. 1–5, doi: 10.1109/PIMRC.2017.8292361.

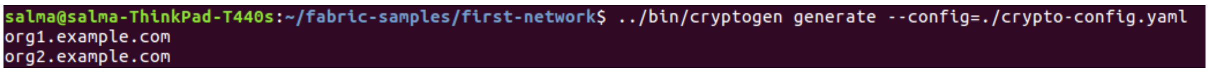
[40] "Introduction | Hyperledger Composer." https://hyperledger.github.io/composer/latest/introduction/introduction (accessed Nov. 03, 2019).

APPENDIX A: MULTI-CHANNEL BLOCKCHAIN IMPLEMENTATION STEPS

This section shows the implementation steps. Initially, when the channels are created, they have the same configuration then, the configuration is changed later using a configuration block that starts at step 8. This update is done to channels 2 and 3 only while channel 1 does not require any changes as its parameters (Batch-timeout and Block Size) are already set to the required values.

1. Generating the cryptographic material

The cryptogen tool consumes crypto-config.yaml file to generate the cryptographic material of the organizations, peers and orderers defined in the file. A file named Crypto-config is created that contains the certificates and the keys of each of these components.

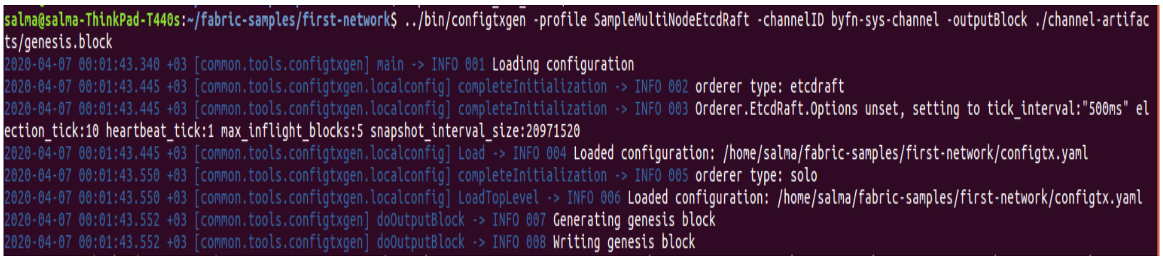


```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/cryptogen generate --config=./crypto-config.yaml
org1.example.com
org2.example.com
```

Figure 25: Generating the Cryptographic material of the organizations

2. Create Raft genesis block

Based on the defined profile in configtx.yaml, the configtxgen tool generates the genesis block for the ordering service. In this case Raft ordering service is used.



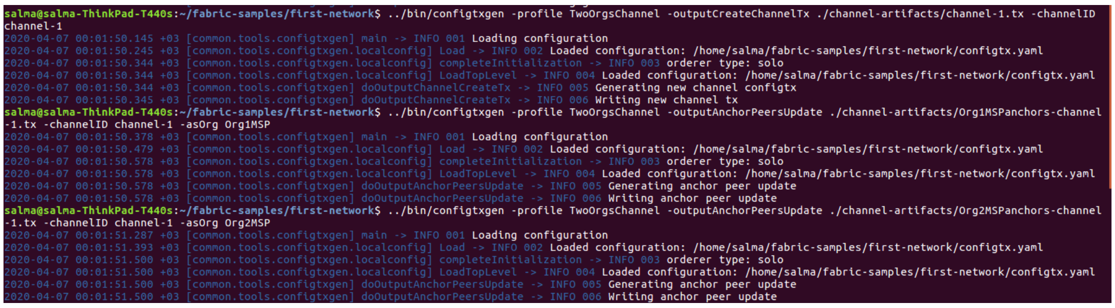
```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile SampleMultiNodeEtcdRaft -channelID byfn-sys-channel -outputBlock ./channel-artifac
ts/genesis.block
2020-04-07 00:01:43.340 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:43.445 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: etcdraft
2020-04-07 00:01:43.445 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" el
ection_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:20971520
2020-04-07 00:01:43.445 +03 [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:43.550 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 005 orderer type: solo
2020-04-07 00:01:43.550 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 006 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:43.552 +03 [common.tools.configtxgen] doOutputBlock -> INFO 007 Generating genesis block
2020-04-07 00:01:43.552 +03 [common.tools.configtxgen] doOutputBlock -> INFO 008 Writing genesis block
```

Figure 26: Creating Raft Genesis Block

3. Creating channels configurations and defining the anchor peers

In this step, the configtxgen tool consumes contfigtx.yaml file to create channels artifacts based on the defined profile (i.e. TwoOrgsChannel in this case). Then, the anchor peer for each organization on the channel is defined. An anchor peer is the peer that allows the organization to communicate with other organizations.

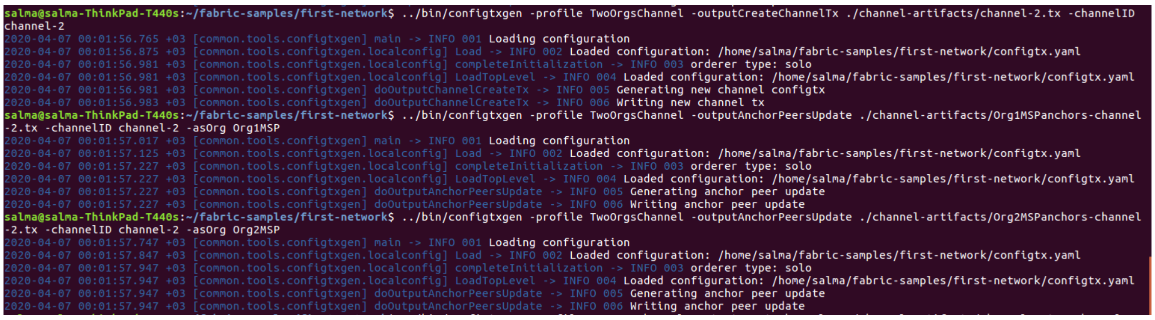
3.1. Channel 1



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel-1.tx -channelID
channel-1
2020-04-07 00:01:50.145 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:50.245 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:50.344 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:01:50.344 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:50.344 +03 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 005 Generating new channel configtx
2020-04-07 00:01:50.345 +03 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 006 Writing new channel tx
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors-channel
-1.tx -channelID channel-1 -asOrg Org1MSP
2020-04-07 00:01:50.378 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:50.479 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:50.578 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:01:50.578 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:50.578 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 005 Generating anchor peer update
2020-04-07 00:01:50.578 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 006 Writing anchor peer update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors-channel
-1.tx -channelID channel-1 -asOrg Org2MSP
2020-04-07 00:01:51.287 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:51.393 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:51.500 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:01:51.500 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:51.500 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 005 Generating anchor peer update
2020-04-07 00:01:51.500 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 006 Writing anchor peer update
```

Figure 27: Channel 1 artifacts creation and defining anchor peers

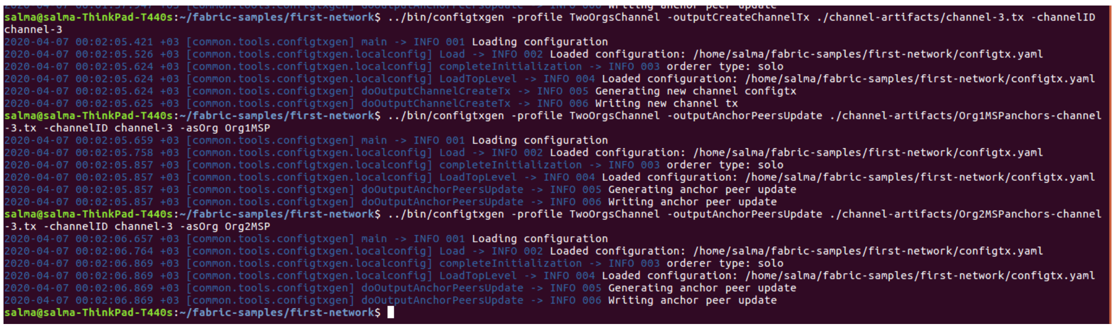
3.2. Channel 2



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel-2.tx -channelID
channel-2
2020-04-07 00:01:56.765 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:56.875 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:56.981 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:01:56.981 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:56.981 +03 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 005 Generating new channel configtx
2020-04-07 00:01:56.983 +03 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 006 Writing new channel tx
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors-channel
-2.tx -channelID channel-2 -asOrg Org1MSP
2020-04-07 00:01:57.017 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:57.125 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:57.227 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:01:57.227 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:57.227 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 005 Generating anchor peer update
2020-04-07 00:01:57.227 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 006 Writing anchor peer update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors-channel
-2.tx -channelID channel-2 -asOrg Org2MSP
2020-04-07 00:01:57.747 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:01:57.847 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:57.947 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:01:57.947 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:01:57.947 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 005 Generating anchor peer update
2020-04-07 00:01:57.947 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 006 Writing anchor peer update
```

Figure 28: Channel 2 artifacts creation and defining anchor peers

3.3. Channel 3

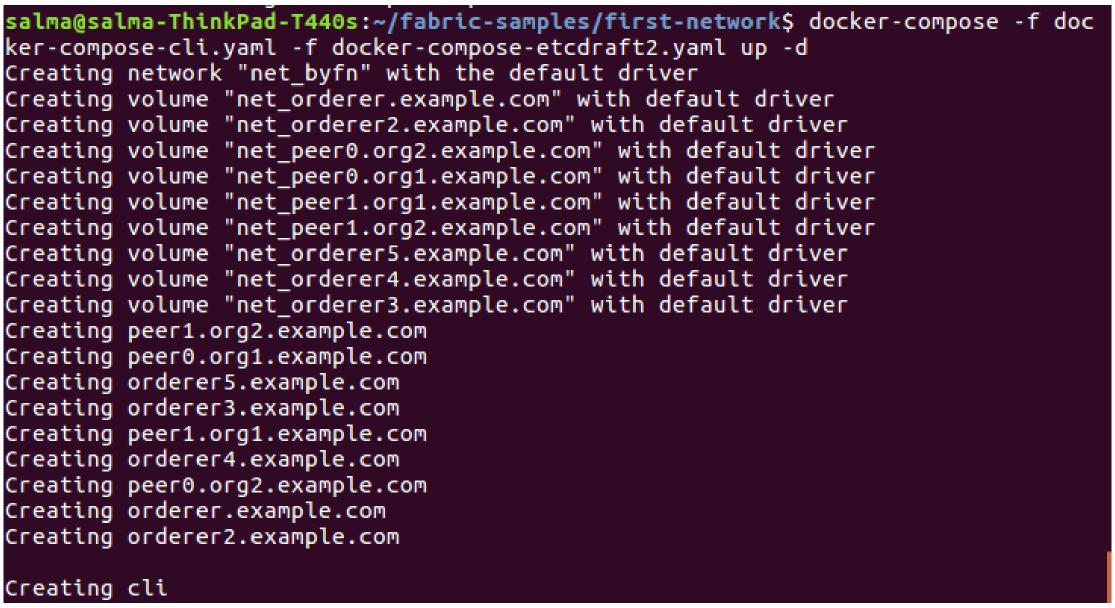


```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel-3.tx -channelID
channel-3
2020-04-07 00:02:05.421 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:02:05.526 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:02:05.624 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:02:05.624 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:02:05.624 +03 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 005 Generating new channel configtx
2020-04-07 00:02:05.625 +03 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 006 Writing new channel tx
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors-channel
-3.tx -channelID channel-3 -asOrg Org1MSP
2020-04-07 00:02:05.659 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:02:05.758 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:02:05.857 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:02:05.857 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:02:05.857 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 005 Generating anchor peer update
2020-04-07 00:02:05.857 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 006 Writing anchor peer update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors-channel
-3.tx -channelID channel-3 -asOrg Org2MSP
2020-04-07 00:02:06.657 +03 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-04-07 00:02:06.764 +03 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:02:06.869 +03 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: solo
2020-04-07 00:02:06.869 +03 [common.tools.configtxgen.localconfig] LoadTopLevel -> INFO 004 Loaded configuration: /home/salma/fabric-samples/first-network/configtx.yaml
2020-04-07 00:02:06.869 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 005 Generating anchor peer update
2020-04-07 00:02:06.869 +03 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 006 Writing anchor peer update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$
```

Figure 29: Channel 3 artifacts creation and defining anchor peers

4. Bring the network up by bringing the docker containers up

In this step we start the network by bringing the docker containers up



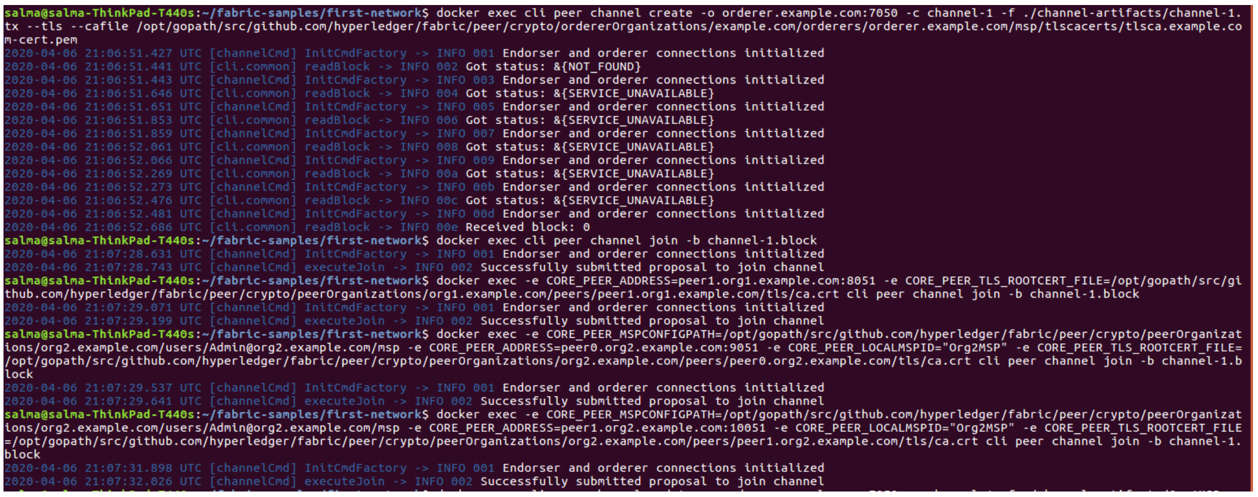
```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker-compose -f doc
ker-compose-cli.yaml -f docker-compose-etcdraft2.yaml up -d
Creating network "net_byfn" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_orderer2.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer1.org1.example.com" with default driver
Creating volume "net_peer1.org2.example.com" with default driver
Creating volume "net_orderer5.example.com" with default driver
Creating volume "net_orderer4.example.com" with default driver
Creating volume "net_orderer3.example.com" with default driver
Creating peer1.org2.example.com
Creating peer0.org1.example.com
Creating orderer5.example.com
Creating orderer3.example.com
Creating peer1.org1.example.com
Creating orderer4.example.com
Creating peer0.org2.example.com
Creating orderer.example.com
Creating orderer2.example.com

Creating cli
```

Figure 30: Bringing the network up

5. Create the channel block for each channel and joining it

In this step, the configurations generated in step 3 are used to create the channels genesis block. Then the peers start joining the channels. Finally, the anchor peers are defined for each organization on the channel.

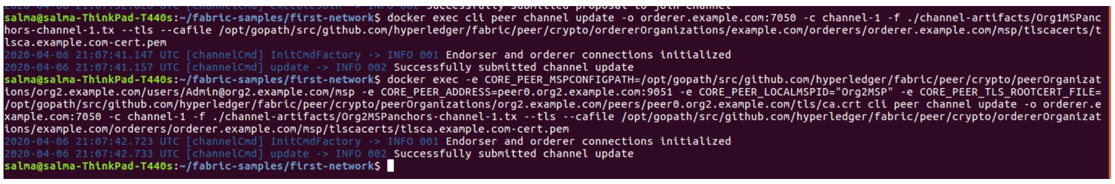
5.1. Creating channel block and the peers join Channel 1



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel create -o orderer.example.com:7050 -c channel-1 -f ./channel-artifacts/channel-1.
tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.co
m-cert.pem
2020-04-06 21:06:51.427 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:06:51.441 UTC [cli.common] readBlock -> INFO 002 Got status: &{NOT_FOUND}
2020-04-06 21:06:51.443 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2020-04-06 21:06:51.646 UTC [cli.common] readBlock -> INFO 004 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:06:51.651 UTC [channelCmd] InitCmdFactory -> INFO 005 Endorser and orderer connections initialized
2020-04-06 21:06:51.853 UTC [cli.common] readBlock -> INFO 006 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:06:51.859 UTC [channelCmd] InitCmdFactory -> INFO 007 Endorser and orderer connections initialized
2020-04-06 21:06:52.061 UTC [cli.common] readBlock -> INFO 008 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:06:52.066 UTC [channelCmd] InitCmdFactory -> INFO 009 Endorser and orderer connections initialized
2020-04-06 21:06:52.269 UTC [cli.common] readBlock -> INFO 00a Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:06:52.273 UTC [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connections initialized
2020-04-06 21:06:52.476 UTC [cli.common] readBlock -> INFO 00c Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:06:52.481 UTC [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
2020-04-06 21:06:52.686 UTC [cli.common] readBlock -> INFO 00e Received block: 0
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel join -b channel-1.block
2020-04-06 21:07:28.631 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:07:28.743 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_ADDRESS=peer1.org1.example.com:8051 -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/gi
thub.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt cli peer channel join -b channel-1.block
2020-04-06 21:07:29.071 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:07:29.199 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel join -b channel-1.b
lock
2020-04-06 21:07:29.537 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:07:29.641 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer1.org2.example.com:10051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel join -b channel-1.
block
2020-04-06 21:07:31.898 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:07:32.026 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
```

Figure 31: Creating channel-1 block and peers join the channel

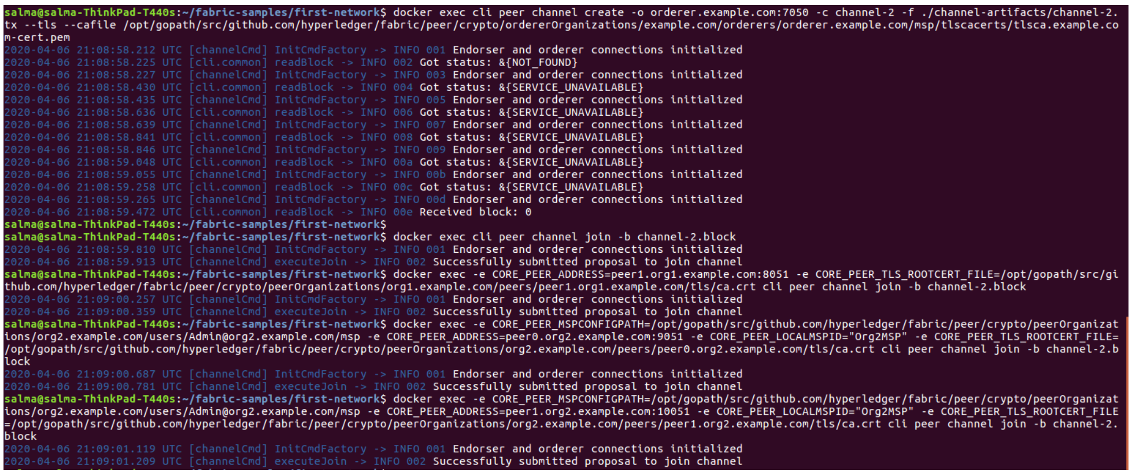
5.2. Updating anchor peers’ definition anchor peers in Channel 1



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel update -o orderer.example.com:7050 -c channel-1 -f ./channel-artifacts/Org1MSPanc
hors-channel-1.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem
2020-04-06 21:07:41.147 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:07:41.157 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel update -o orderer.e
xample.com:7050 -c channel-1 -f ./channel-artifacts/Org2MSPanchors-channel-1.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizat
ions/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
2020-04-06 21:07:42.723 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:07:42.733 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ 
```

Figure 32: Updating anchor peers’ definition inn channel 1

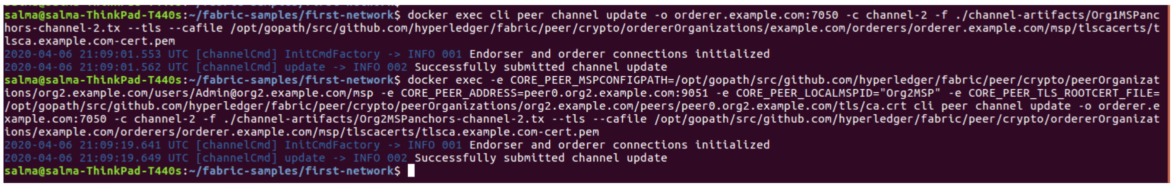
5.3. Creating channel block and the peers join Channel 2



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel create -o orderer.example.com:7050 -c channel-2 -f ./channel-artifacts/channel-2.
tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.co
m-cert.pem
2020-04-06 21:08:58.212 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:08:58.225 UTC [cli.common] readBlock -> INFO 002 Got status: &{NOT_FOUND}
2020-04-06 21:08:58.227 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2020-04-06 21:08:58.430 UTC [cli.common] readBlock -> INFO 004 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:08:58.435 UTC [channelCmd] InitCmdFactory -> INFO 005 Endorser and orderer connections initialized
2020-04-06 21:08:58.636 UTC [cli.common] readBlock -> INFO 006 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:08:58.639 UTC [channelCmd] InitCmdFactory -> INFO 007 Endorser and orderer connections initialized
2020-04-06 21:08:58.841 UTC [cli.common] readBlock -> INFO 008 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:08:58.846 UTC [channelCmd] InitCmdFactory -> INFO 009 Endorser and orderer connections initialized
2020-04-06 21:08:59.048 UTC [cli.common] readBlock -> INFO 00a Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:08:59.055 UTC [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connections initialized
2020-04-06 21:08:59.258 UTC [cli.common] readBlock -> INFO 00c Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:08:59.265 UTC [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
2020-04-06 21:08:59.472 UTC [cli.common] readBlock -> INFO 00e Received block: 0
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel join -b channel-2.block
2020-04-06 21:08:59.810 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:08:59.913 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_ADDRESS=peer1.org1.example.com:8051 -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/gi
thub.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt cli peer channel join -b channel-2.block
2020-04-06 21:09:00.257 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:09:00.359 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel join -b channel-2.b
lock
2020-04-06 21:09:00.687 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:09:00.781 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer1.org2.example.com:10051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel join -b channel-2.
block
2020-04-06 21:09:01.119 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:09:01.209 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
```

Figure 33: Creating channel 2 block and peers join the channel

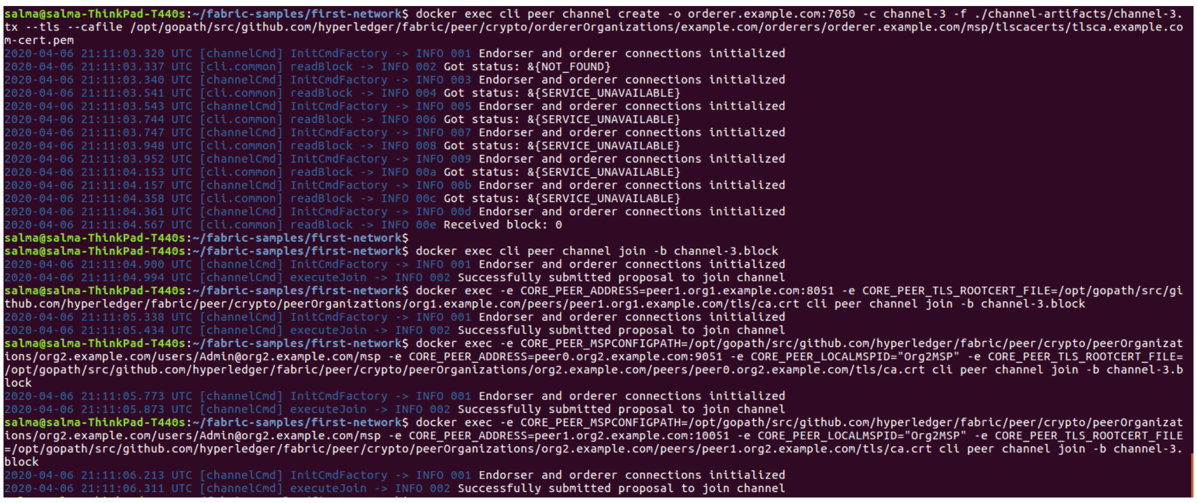
5.4. Updating anchor peers’ definition anchor peers in Channel 2



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel update -o orderer.example.com:7050 -c channel-2 -f ./channel-artifacts/Org1MSPanc
hors-channel-2.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem
2020-04-06 21:09:01.553 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:09:01.562 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel update -o orderer.e
xample.com:7050 -c channel-2 -f ./channel-artifacts/Org2MSPanchors-channel-2.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizat
ions/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
2020-04-06 21:09:19.641 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:09:19.649 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ 
```

Figure 34: Updating anchor peers’ definition in channel 2

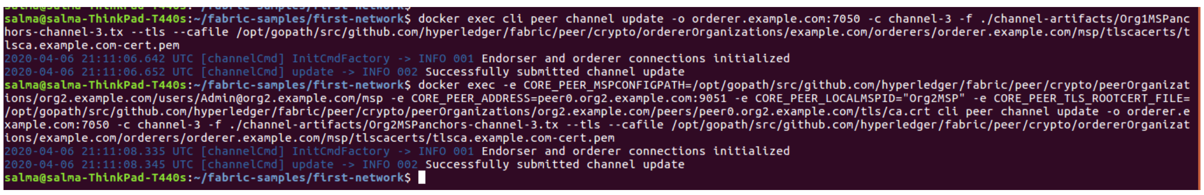
5.5. Creating channel block and the peers join Channel 3



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel create -o orderer.example.com:7050 -c channel-3 -f ./channel-artifacts/channel-3.
tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.co
m-cert.pem
2020-04-06 21:11:03.320 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:03.337 UTC [cli.common] readBlock -> INFO 002 Got status: &{NOT_FOUND}
2020-04-06 21:11:03.340 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2020-04-06 21:11:03.541 UTC [cli.common] readBlock -> INFO 004 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:11:03.543 UTC [channelCmd] InitCmdFactory -> INFO 005 Endorser and orderer connections initialized
2020-04-06 21:11:03.744 UTC [cli.common] readBlock -> INFO 006 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:11:03.747 UTC [channelCmd] InitCmdFactory -> INFO 007 Endorser and orderer connections initialized
2020-04-06 21:11:03.948 UTC [cli.common] readBlock -> INFO 008 Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:11:03.952 UTC [channelCmd] InitCmdFactory -> INFO 009 Endorser and orderer connections initialized
2020-04-06 21:11:04.153 UTC [cli.common] readBlock -> INFO 00a Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:11:04.157 UTC [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connections initialized
2020-04-06 21:11:04.358 UTC [cli.common] readBlock -> INFO 00c Got status: &{SERVICE_UNAVAILABLE}
2020-04-06 21:11:04.361 UTC [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
2020-04-06 21:11:04.567 UTC [cli.common] readBlock -> INFO 00e Received block: 0
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel join -b channel-3.block
2020-04-06 21:11:04.900 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:04.994 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_ADDRESS=peer1.org1.example.com:8051 -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/gi
thub.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt cli peer channel join -b channel-3.block
2020-04-06 21:11:05.338 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:05.434 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel join -b channel-3.b
lock
2020-04-06 21:11:05.773 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:05.873 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer1.org2.example.com:10051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel join -b channel-3.
block
2020-04-06 21:11:06.213 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:06.311 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
```

Figure 35: Creating channel 3 block and peers join the channel

5.6. Updating anchor peers' definition anchor peers in Channel 3

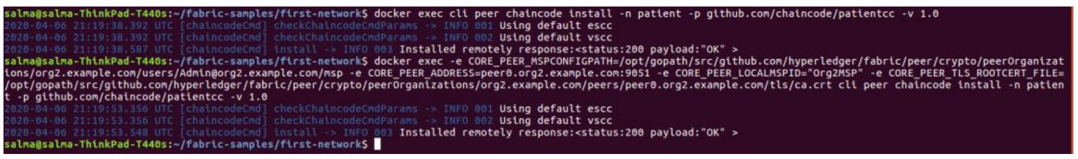


```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer channel update -o orderer.example.com:7050 -c channel-3 -f ./channel-artifacts/Org1MSPanc
hors-channel-3.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem
2020-04-06 21:11:06.642 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:06.652 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer channel update -o orderer.e
xample.com:7050 -c channel-3 -f ./channel-artifacts/Org2MSPanchors-channel-3.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizat
ions/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
2020-04-06 21:11:08.335 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-04-06 21:11:08.345 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$
```

Figure 36: Updating anchor peers' definition in channel 3

6. Install the chaincode (patientCC) on the peers

In this step the chaincode is installed on the peers, this step is done only once for the peers that will be using the chaincode



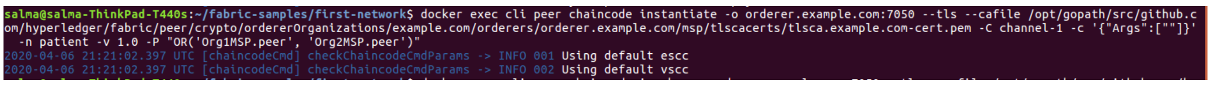
```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer chaincode install -n patient -p github.com/chaincode/patientcc -v 1.0
2020-04-06 21:19:38.392 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-04-06 21:19:38.392 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2020-04-06 21:19:38.587 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e 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 cli peer chaincode install -n patien
t -p github.com/chaincode/patientcc -v 1.0
2020-04-06 21:19:53.356 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-04-06 21:19:53.356 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2020-04-06 21:19:53.548 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$
```

Figure 37: Installing the chaincode on the peers

7. Instantiate the chaincode on each channel

Unlike the installing that is done per peer, the instantiation of the chaincode is done per channel, each channel that will be using the chaincode should have a copy of it.

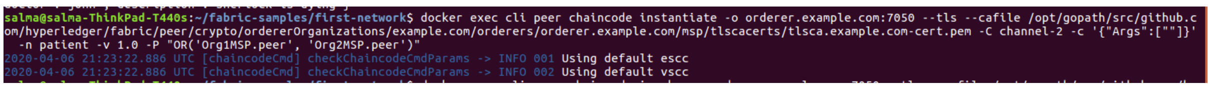
7.1. Instantiating the patient chaincode on channel 1



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.c
om/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C channel-1 -c '{"Args":[""]}'
  -n patient -v 1.0 -P "OR('Org1MSP.peer', 'Org2MSP.peer')"
2020-04-06 21:21:02.397 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-04-06 21:21:02.397 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
```

Figure 38: Instantiating the chaincode on channel 1

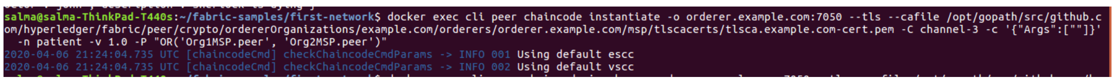
7.2. Instantiating the patient chaincode on channel 2



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.c
om/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C channel-2 -c '{"Args":[""]}'
  -n patient -v 1.0 -P "OR('Org1MSP.peer', 'Org2MSP.peer')"
2020-04-06 21:23:22.886 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-04-06 21:23:22.886 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
```

Figure 39: Instantiating the chaincode on channel 2

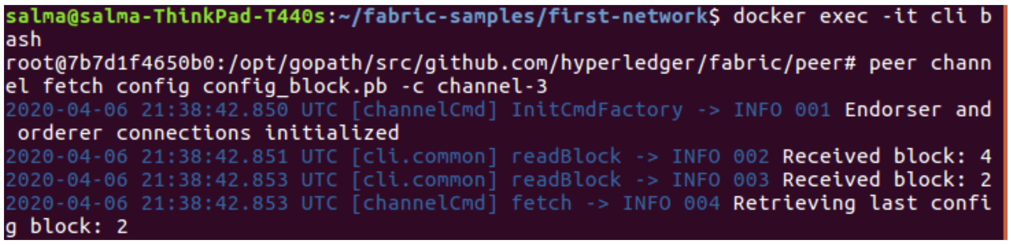
7.3. Instantiating the patient chaincode on channel 3



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec cli peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.c
om/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C channel-3 -c '{"Args":[""]}'
  -n patient -v 1.0 -P "OR('Org1MSP.peer', 'Org2MSP.peer')"
2020-04-06 21:24:04.735 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-04-06 21:24:04.735 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
```

Figure 40: Instantiating the chaincode on channel 3

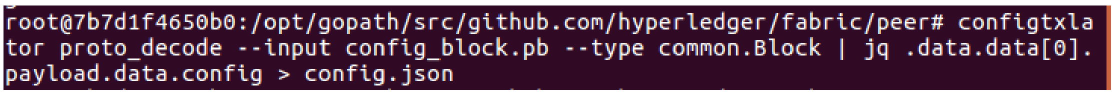
8. Fetching the current channel configuration of the channels in portobuf format



```
salma@salma-ThinkPad-T440s:~/fabric-samples/first-network$ docker exec -it cli b
ash
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chann
el fetch config config_block.pb -c channel-3
2020-04-06 21:38:42.850 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and
 orderer connections initialized
2020-04-06 21:38:42.851 UTC [cli.common] readBlock -> INFO 002 Received block: 4
2020-04-06 21:38:42.853 UTC [cli.common] readBlock -> INFO 003 Received block: 2
2020-04-06 21:38:42.853 UTC [channelCmd] fetch -> INFO 004 Retrieving last confi
g block: 2
```

Figure 41: Fetching the current configuration of channel 3

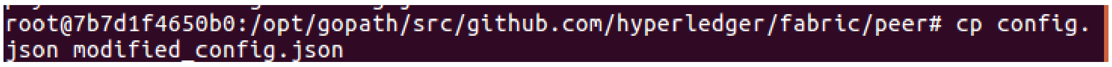
9. Translate the portobuf format to readable JSON format



```
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# configtxla
tor proto_decode --input config_block.pb --type common.Block | jq .data.data[0].
payload.data.config > config.json
```

Figure 42: Convert the portobuf format to readable JSON format

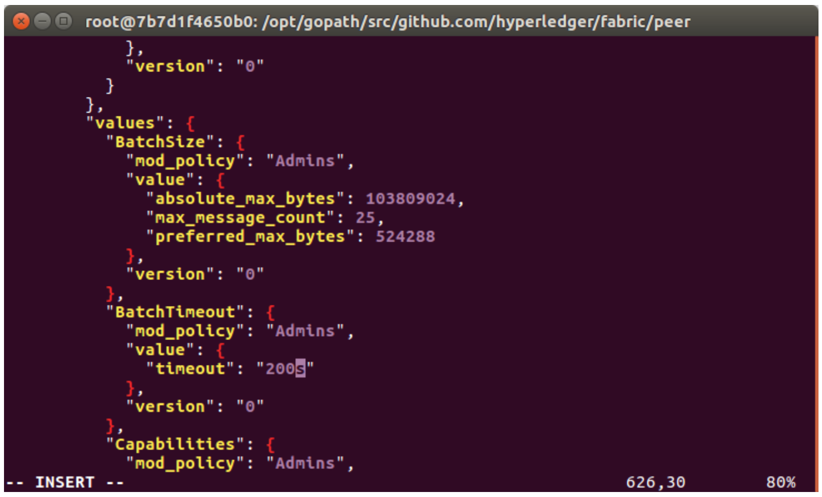
10. Create a new copy of the json file to apply the modifications



```
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# cp config.
json modified_config.json
```

Figure 43: create a new copy of the config file

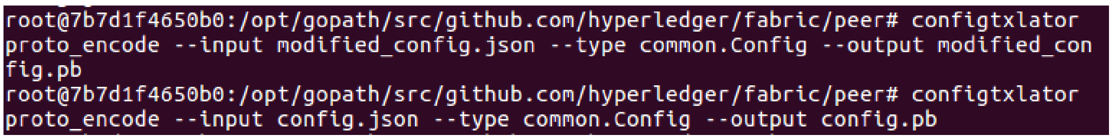
11. Modify the configurations on the new copy of the JSON file



```
root@7b7d1f4650b0: /opt/gopath/src/github.com/hyperledger/fabric/peer
            },
            "version": "0"
          }
        },
        "values": {
          "BatchSize": {
            "mod_policy": "Admins",
            "value": {
              "absolute_max_bytes": 103809024,
              "max_message_count": 25,
              "preferred_max_bytes": 524288
            },
            "version": "0"
          },
          "BatchTimeout": {
            "mod_policy": "Admins",
            "value": {
              "timeout": "200s"
            },
            "version": "0"
          },
          "Capabilities": {
            "mod_policy": "Admins",
-- INSERT --                                               626,30        80%
```

Figure 44: Applying the modifications

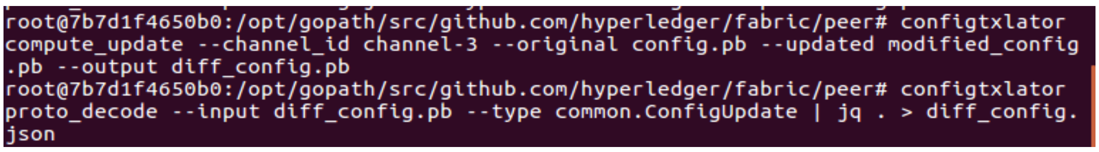
12. Convert the old and the modified JSON files to portobuf format



```
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# configtxlator
proto_encode --input modified_config.json --type common.Config --output modified_con
fig.pb
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# configtxlator
proto_encode --input config.json --type common.Config --output config.pb
```

Figure 45: converting teh modified JSON file to portobuf format

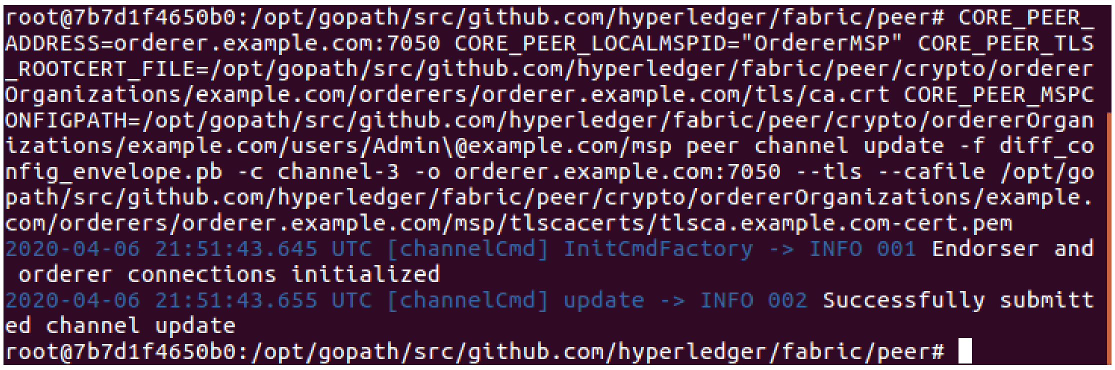
13. Compute the difference between the old portobuf file and the new one, then convert the difference to JSON format



```
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# configtxlator
compute_update --channel_id channel-3 --original config.pb --updated modified_config
.pb --output diff_config.pb
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# configtxlator
proto_decode --input diff_config.pb --type common.ConfigUpdate | jq . > diff_config.
json
```

Figure 46: Computing teh difference and converting it to JSON

14. Sign the updates by the orderer and send the channel update transaction to the ledger



```
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_
ADDRESS=orderer.example.com:7050 CORE_PEER_LOCALMSPID="OrdererMSP" CORE_PEER_TLS
_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer
Organizations/example.com/orderers/orderer.example.com/tls/ca.crt CORE_PEER_MSPC
ONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrgan
izations/example.com/users/Admin\@example.com/msp peer channel update -f diff_co
nfig_envelope.pb -c channel-3 -o orderer.example.com:7050 --tls --cafile /opt/go
path/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.
com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
2020-04-06 21:51:43.645 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and
 orderer connections initialized
2020-04-06 21:51:43.655 UTC [channelCmd] update -> INFO 002 Successfully submitt
ed channel update
root@7b7d1f4650b0:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Figure 47: Signing the updates and sending the update transaction