Performance and Scalability of Blockchain Networks and Smart Contracts

Mattias Scherer

Mattias Scherer
Spring 2017
Degree Project in Computing Science and Engineering, 30 ECTS Credits
Supervisor: Jerry Eriksson, Umeå University
External Supervisor: Oskar Janson, Cinnober Financial Technology
Examiner: Henrik Björklund, Umeå University
Master of Science Programme in Computing Science and Engineering, 300 ECTS Credits

Abstract

The blockchain technology started as the innovation that powered the cryptocurrency Bitcoin. But in recent years, leaders in finance, banking, and many more companies has given this new innovation more attention than ever before. They seek a new technology to replace their system which are often inefficient and costly to operate. However, one of the reasons why it not possible to use a blockchain right away is because of the poor performance. Public blockchains, where anyone can participate, can only process a couple of transaction per second and is therefore far from usable in the world of finance. Permissioned blockchains is another type of blockchain where only a restricted set of users have the rights to decide what will be recorded in the blockchain. This allows permissioned blockchains to have a number of advantages over public blockchains. Most notably is the ability to split the network into segments where only a subset of nodes needs to validate transactions to a particular application, allowing the use of parallel computing and better scaling. Moreover, the validating nodes can be trusted, allowing the use of consensus algorithm which offer much more throughput.

In this paper, we compare public blockchain with permissioned blockchain and address the notable trade-offs: decentralization, scalability and security, in the different blockchain networks. Furthermore, we examine the potential of using a permissioned blockchain to replace the old systems used in financial institutes and banks by launching a Hyperledger Fabric network and run stress tests.

It is apparent that with less decentralization, the performance and scalability of Hyperledger Fabric network is improved and it is feasible that permissioned blockchain can be used in finance.

Acknowledgements

First and foremost, I want to thank my supervisor at Cinnober Financial Technology, Oskar Janson, for sharing his great knowledge. I would also like to thank Jerry Eriksson for providing excellent feedback.

Contents

1 Introduction

1.1 Background

Although still very much in an early stage, blockchain and distributed ledger solutions are said to be able to transform the current financial infrastructure, especially the post-trade side. Most notable of all applications of blockchain technology is Bitcoin, which is an open decentralized network with an effectively immutable database of transactions, shared by all full nodes in the network. The internal currency, bitcoins, is provided to miners who help secure the network by participating in a computational-expensive consensus algorithm.

Since the beginning of Bitcoin, other similar solutions have appeared. Some are modified forks of Bitcoin while some are built from scratch using the same ideas. Ethereum is a network that expands the Bitcoin scripting language to be Turing complete with the hope of enabling more complex applications to be run and verified by the nodes in the network[1]. These applications are commonly known as Smart Contracts and the increased flexibility of the scripting language is expected to enable a large number of uses in finance. The Hyperledger project is an open source collaborative effort created to advance cross-industry blockchain technologies[2]. It is a global collaboration including leaders in finance, banking, Internet of Things, supply chain, manufacturing and technology. Fabric is an implementation of blockchain technology that is intended as a foundation for developing blockchain applications or solutions. It offers a modular architecture allowing components, such as consensus and membership services, to be plug-and-play. It leverage container technology to host smart contracts called ”chaincode” that comprises the application logic of the system[3].

With a more complex solution comes many additional challenges. For example, if no special techniques are applied, every node in the network will verify every transaction. This puts a limit on how many smart contracts can be processed in each block, and in turn how large a smart contract application can be before it affects the overall performance of the network too much. If we want to take functions provided centrally today, for example by exchanges and clearing houses, and put them in a blockchain network, it is necessary that we find ways to remove or handle these limitations. It is also necessary that we are aware of the current limits in performance when creating smart contracts when evaluating potential uses of applications run in a blockchain network.

1.2 Thesis goals

The purpose of this thesis is to evaluate different blockchain networks with focus on performance and scalability. We will examine what scalability issues Bitcoin will eventually have to face as well as Ethereum’s interesting solutions to their scalability issues. Meanwhile other types of blockchain networks which are more centralized, e.g. permissioned

networks, do not have the same scalability issues as public network. However, this means that permissioned networks will not have the same degree of decentralization as a public network would have. This dilemma we are examining now reminds us of the CAP theorem, which states that it is impossible for a distributed computer system to simultaneously provide more than two out of three of the following guarantees: Consistency, Availability, and Partition tolerance. In the sense of a blockchain network, this translates to: Decentralization, Scalability, and Security. Since this technology is young we are still learning and trying out new techniques that could provide a blockchain network that is decentralized, scalable, and secure.

More specifically, this paper aims to:

- Provide an analysis of the benefits and drawbacks of Bitcoin, Ethereum and Hyperledger Fabric in regards to performance and scalability, and address the three-way trade-off between decentralization, scalability and security
- Address the possible requirements needed for applications that will run on a blockchain network
- Outline where performance constraint can exist in Hyperledger Fabric when tested with use cases found within the finance industry, and how does different network configuration affect the system's performance
- Outline current techniques to improve performance on applications running on Bitcoin or Ethereum
- Address whether these techniques could improve the performance of Hyperledger Fabric
- Address the limitations of blockchain networks have compared to traditional centralized solutions in regards to performance

2 Blockchain technology

A blockchain is a special kind of distributed database. A distributed works well when all entities trust each other and do not want to keep duplicate records of the same data. Blockchain however, comes into play when the entities cannot trust each other, that is, there is no single entity in control and we need a magical database that is: Distributed and decentralized. A block contains transactional data and they can be thought of as a page of a ledger. Each block also contains a timestamp and a hashed link to a previous block, creating a *chain* of *blocks*. By design, blockchains are resistant to modification of data; once a new entry has been recorded, the data in that block cannot be altered retroactively[4]. Moreover, older blocks cannot be altered without breaking the chain to every block that is recorded afterwards. If an attacker would attempt to modify a block, he would have to change all blocks that happened afterwards to the most recent block. This attack is very difficult to achieve and will be explored in more detail later in this chapter.

The true utility of the blockchain is through the use of a peer-to-peer network. When the blockchain is distributed over a network of nodes, each node can validate the actions of other nodes, as well as the ability to create, authenticate and verify new transaction to be recorded onto the blockchain. The network itself incentives the nodes to enforce the protocol or the law of the blockchain for all other nodes by rewarding those who do it right, and ignoring those who do it wrong. Invalid data submitted by a node will be discarded by all other nodes and will not be recorded onto the blockchain.

The Blockchain can be thought of as a single shared truth and it is secure by design. The nodes run the same software and manage the same data, therefore, they can fail without consequences. It is an example of a distributed computing system with high Byzantine fault tolerance that enables trustless consensus. This makes blockchain suitable for the recording of events, digital assets and stocks, cryptocurrency, voting system, etc., without being controlled by powerful forces.

2.1 Merkle tree

A Merkle tree is a way of hashing a larger chunk of data into a single hash (Figure 2.1.1). A block holds batches of valid transactions that are hashed and encoded into a merkle tree. Merkle trees are a fundamental component in the blockchain technology. With it, each block in the blockchain contains a summary of all the transactions in the block. Although it is theoretically possible to create a huge block that directly contains every transaction in the header. But doing so poses large scalability challenges that arguably puts the use of blockchains out of reach of all but the most powerful computers. Merkle trees allow small and simple smart phones, laptop and even internet of things device, to powerful computers to run a blockchain.

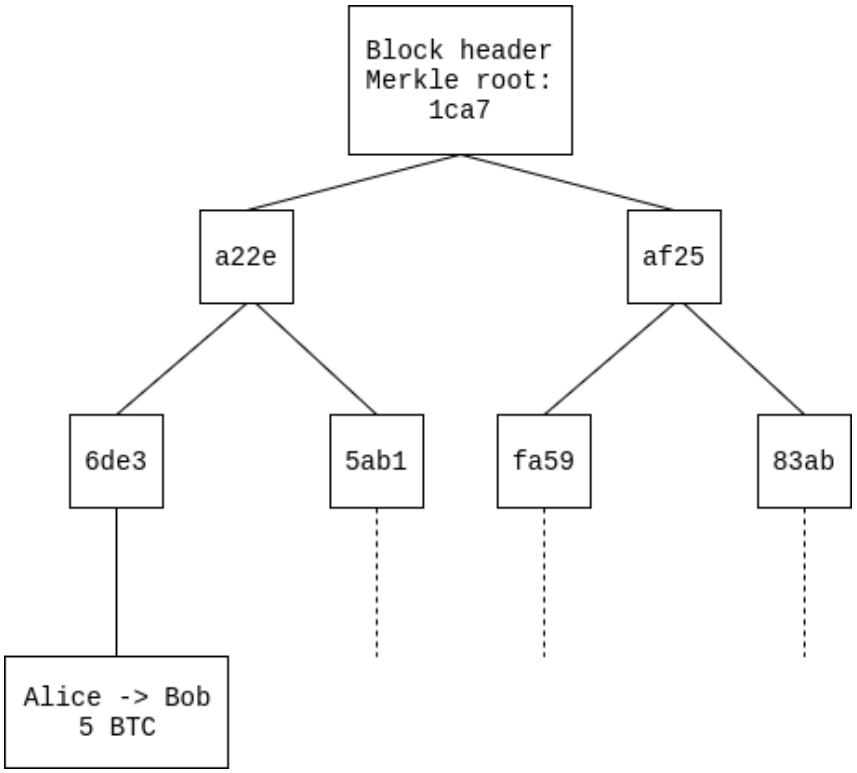

Figure 2.1.1: An illustration of a Merkle tree

The tree is constructed bottom-up. The leaves in the tree are hashed transactions from a single block. Consecutive pairs of leaf nodes are then summarized in a parent node, by concatenating the two hashes and hashes them together, until there is only one hash, called the merkle root. Then using a neat mechanism known as Merkle proofs, devices can download the chain of block header, 80-byte chunk of data in bitcoin for each block, and verify that a certain transaction is recorded in the blockchain. A *light client* is a client that can only download the chain of block headers and they can verify the hashing for a branch is consistent going all the way up the tree, and therefore the given transaction is at that position in the tree[5].

2.2 Bitcoin: A Peer-to-Peer Electronic Cash System

The first blockchain was conceptualized by an unknown author under the pseudonym Satoshi Nakamoto in 2008 with the publication of the white paper *Bitcoin: A Peer-to-Peer Electronic Cash System*[6]. The Bitcoin network combines several cryptographic components to create a Peer-to-Peer(P2P) payment system today known as cryptocurrency, which allows for true peer-to-peer exchange of value without the need of a trusted intermediary.

Bitcoin transactions are sent from and to Bitcoin wallets, and are digitally signed using private keys. The digital bitcoin does not exist at one's account in the same way dollars are held in a bank account, only records of bitcoin transaction exists. Bitcoin wallets store the private key that is required to access your bitcoin from a transaction and to spend your funds. This model is known as a UTXO which is an abbreviation for Unspent Transaction Output,

meaning, transaction in which bitcoins were transferred to the user that has not yet been spent. All input to a transaction must be one or more UTXO available in the blockchain for it to be valid. It is possible that inputs can be invalid if either the transaction is trying to double-spend some bitcoin that were already spent or the transaction is trying to spend bitcoin that does not exist. To know a key in this sense is analogous to owning bitcoins, and loosing a private key essentially means that the bitcoin on that key is gone forever.

A user sends bitcoins by creating a bitcoin transaction and selects a set of UTXO as input and the amount of bitcoin the user wishes to transfer to each party as output. Transactions can also have multiple outputs to pay multiple parties. Also, if the combined amount of bitcoins in the input is more than the output, another output can be used to send bitcoins back to the user. If the input is worth 10 BTC but the user wants to send only 5 BTC, Bitcoin will create two outputs worth 10 BTC: one to the destination, and one back to the user (Figure 2.2.1). Any remaining input bitcoins not redeemed in an output is considered a transaction fee; whoever generates the block will get it. A valid transaction must have a greater or equal amount of bitcoins in the input as in the output. The output is addressed using the bitcoin address, which is in this sense is equal to the public key. The user then signs the transaction with the private key associated with the UTXO's and the transaction is then broadcasted to the Bitcoin network.

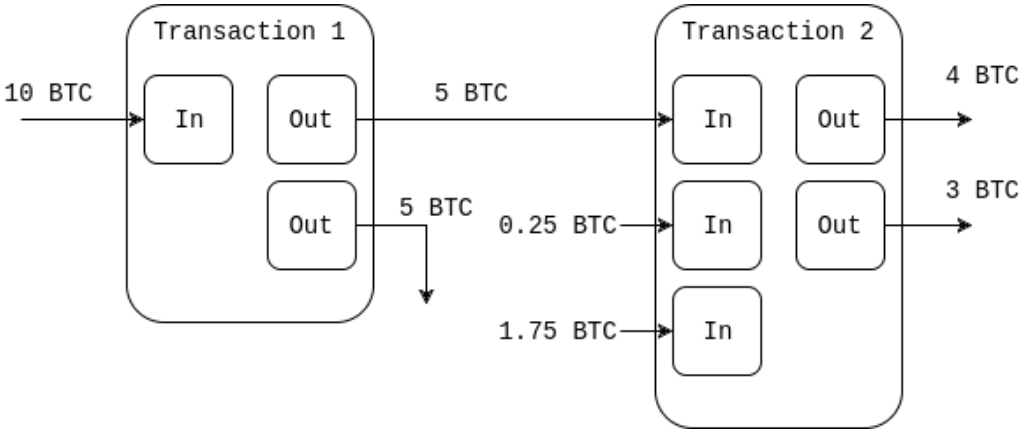

Figure 2.2.1: Bitcoin transaction using output from other transaction as input

2.2.1 Consensus model

The transaction recorded in a block is decided by the *miner* that generated the block. A miner that generated a block receive the cumulative amount of fees from transaction as well as a special transaction that is always the first transaction appearing in every block, known as a block reward, that they can credit to their own wallet. The block reward transaction is how new bitcoins are created in the system. In order to generate a new block and claim this reward, miners must compete to solve a difficult mathematical puzzle, an answer that is unique to each block, also known as a *nonce*. A block cannot be submitted to the blockchain without the correct answer. Having the correct answer is what is known as the Proof-of-Work that creates a distributed trustless consensus and solves the double-spend problem[6]. Proof-of-Work is a piece of data which is costly and time consuming to produce but easy for others to verify. To find the correct answer, a miner must hash to a value less than the current target. The difficulty of this work is adjusted to limit the rate at which a new block can be

generated by the network to one block every ten minutes. The networks hash rate is what dictates if the difficulty has to increase. The more miners that join the Bitcoin network, the higher the network hash rate is. Due to the very low probability of successful generation, this makes it unpredictable which miner in the network will be able to generate the next block. This creates an arguably fair distribution of Bitcoin. Each block contains the hash of the preceding block, thus, the blockchain contains a immense amount of work. Changing a block by making a new block referencing the same predecessor requires regeneration of all successor and redoing the immense work they contain. This protects the blockchain from tampering.

Since it takes time for a newly generated block to be propagated through the network, there is a probability that another miner have found the right answer and generated a block as well. Each miner will mine onto which ever block they received first, creating a fork in the blockchain (Figure 2.2.2). Honest miners only build onto the *longest valid chain* since the block reward and transaction fees are only redeemable if the block is part of the longest chain. The probability that the divided miners on the separate chains continue to solve blocks simultaneously diminishes with each mined block, to the point where one chain eventually overtakes the other chain, and that chain is dropped. Miners would not risk trying to solve a block that may not be a part of the longest chain. This creates an incentive for miners to work towards a single version of the blockchain.

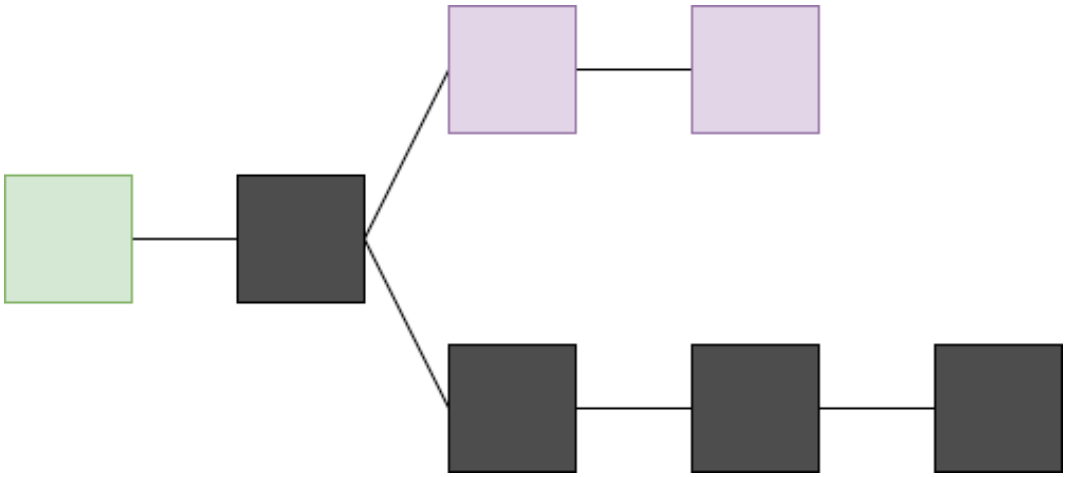

Figure 2.2.2: The purple blocks are ”orphan” blocks while the black blocks are the longest validated blockchain. The green block is called the ”genesis block”, the first block in the blockchain.

When blocks are not part of the conventional blockchain they are known as *orphaned blocks*. It means that the blocks were successfully mined but are not included on the current longest chain, likely because some other block at the same height had it’s chain extended first. Orphan block could easily be confused with *stale blocks*, which are blocks that already have a valid successor[7][8]. In the graph above, all black blocks are stale except the most recent one. Mining on a stale block is considered bad since it is mining on old data, which will be a waste of energy and money because the block will not be included in the blockchain.

Transactions that are broadcast to the network are collected by the miners and added to the block they are trying to solve. When a new block is generated with correct answer, the newly created block is propagated into the network containing a set of transaction which

the miner has collected. A transaction fee is necessary to include because miners prioritize transaction with higher fees to be recorded in the next block since not all transaction in the pool of transaction might not fit inside a block. The block rewards is an incentive for miners to continue generating blocks and the transaction fee is an incentive for the miner to include the transaction in their block. In the future, as the amount of new bitcoins created from a new bitcoins decreases, the fees will make up a much more important percentage of mining income to secure the network. The block reward started at 50 BTC in the first block and halves every 210,000 blocks. This means that block number 210,001 and onward rewards 25 BTC. Since blocks are mined on average every ten minutes[9], 144 blocks are generated each day on average. At 144 blocks per day, 210,000 blocks take on average four years to mine. The block reward will continue to be halved until a total of 21 million Bitcoins are produced, then the block reward will stop and no more Bitcoins will ever be produced.

The Proof-of-Work of Bitcoin allows anyone with a processing unit to participate in the competition of creating new blocks. It introduces bitcoins into the network in a steady pace using an arguably fair distribution mechanism. It makes the blockchain tamper-resistant due to the immense work needed to change a block and allows for a consensus mechanism that is resistant to Sybil attacks. Although the Proof-of-Work of Bitcoin accomplishes these tasks, they are still vulnerable to certain attacks. Since all miners in the networking are continuously completing new Proof-of-Work and try to always generate block on the longest chain because it is in their self-interest. However, an attacker could manipulate the public ledger at will by controlling $>50\%$ of the hash rate, commonly known as a 51% attack or majority hash rate attack. The attacker would not have complete control over the network[10].

There is only a couple of things that can be achieved with 51% of the network hash rate. The attacker could prevent transactions of their choosing from gaining any confirmation, thus preventing users from sending bitcoins between addresses. A confirmation is when a transaction is verified and recorded in a block that is included the the blockchain[11]. When potential loss due to spending is high, as most exchanges and other merchant, requires the transaction to be six blocks or more deep before the transaction is confirmed. For inexpensive or non-fungible items, it is common to complete the exchange as soon as the transaction is seen on the network. The attacker could reverse transaction they send during the time they are in control, allowing for double-spend transactions. In reality a 51% attack from a small group is impossible considering the hash rate of the network is at the time of writing 2,989,368 TH/s(Tera hash per second) or 37,964,981 PetaFLOPS/s[12]. Comparing with fastest supercomputer in the world, the Sunway TaihuLight, with 125 PetaFLOPS/s[13]. The comparison with the supercomputer is not really fair since the mining network uses ASICs(Application Specific Integrated Circuits)[14] that are much more efficient than general computing circuits, but it gives an idea of how powerful the mining network is. If an attacker wants to control more than 50 percent of the hash rate, the attacker has to purchase huge amount of ASICs to compete with the current hash rate. Antminer S9 is the most cost-effective mining hardware with a cost of around $2,400 for a hash rate of 14 TH/s[15], and this is not counting the cost of electricity. The price to compete with the current hash rate for a small group of attackers is too much. However, today's mining is done in groups of cooperating miners who agree to share the block reward and transaction fees, these group are known as mining pools. A 51% attack is feasible if the four biggest mining pools collaborate in the attack[16].

2.3 Ethereum

Ethereum is an open source platform to build and distribute the next generation of decentralized applications[17][18]. Applications where there is no middleman, where user interact with social systems, financial systems, gaming interfaces, all in peer-to-peer fashion. Ethereum provide a blockchain with a built-in Turing-complete programming language that can be used to create *smart contracts* and to encode arbitrary state transition functions, allowing users to create any of the system described above, as well as other applications not yet imagined. Simply put, Ethereum takes the core blockchain technology that Bitcoin conceptualized and evolves it. Everything Bitcoin can do, store money, sending/receiving payments, Ethereum can also do but does it more efficiently.

2.3.1 Smart contracts

Smart contracts are programs written in Solidity[19], a smart contract specific language, which compiles to EVM (Ethereum Virtual Machine) opcodes. Contracts are defined by their creators, but their execution, and by extension the services they offer, is provided by the Ethereum network itself. They exist in the network and can be executable as long as the network itself continues to exist, and will only disappear if they are programmed to self destruct. Solidity provides a more expressive and complete language than Bitcoin for scripting. It is a Turing complete language capable of holding objects on the Ethereum blockchain, interact with other contracts, make decisions, store data, and send ether to others. Ether is the necessary element that is used as fuel for operating application on Ethereum. It is a form of payment made by clients to the machines executing the requested operations[20].

2.3.2 EVM

The EVM can be thought of as a distributed global computer where all smart contracts are executed. Given that every smart contract in the network is executed on every machine, there has to be a mechanism to limit the amount of resources a smart contract can use. Ethereum uses two additional fields in a transaction known as `STARTGAS` and `GASPRICE`. `STARTGAS` is value representing the maximum number of computational steps the transaction execution is allowed to take. `GASPRICE` is a value representing the fee the sender pays per computational step, this value is the amount of ether per gas the sender is prepared to pay. For example, if A sends a transaction to B with 2000 gas and 0.001 ether gasprice, 2000 * 0.001 = 2 ether, then 2 ether is subtracted from A's account. Although it might not be necessary to use up all the gas. If the operation on a contract finishes with some gas remaining, what is left is computed back to ether using the gasprice and sent back to A's account. If the sender did not have enough ether, or the code execution ran out of gas, revert all state changes except for the payment of the fees. Most computational step costs one gas, the cost varies depends on how expensive the computation is or increased amount of data to be stored as part of the state. There is also a fee of five gas for every byte in the transaction data. The `STARTGAS` and `GASPRICE` fields are crucial for Ethereum's anti-denial of service model. These two values prevent accidental or hostile infinite loops or other computational wastage in code. The intent of this system is to require the attacker to pay proportionately for every resource they consume, including computation, bandwidth and storage.

2.3.3 Accounts

In Bitcoin, user's balance are stored in the structure based on UTXO described earlier, whereas the state of Ethereum is made up of objects called "accounts". Each account having a 20-byte address and state transition being direct transfer of value and information between accounts. An account can be divided into two types: externally owned accounts, controlled by user with their private key, and contract accounts, controlled by their code. An Ethereum account contains four fields:

- The nonce, a counter to make sure each transaction can only be used once.
- The account's current ether balance
- The account's contract code, if it is a contract account
- The account's storage

An externally owned account has no code, and one can send messages to other externally owned accounts or contract account by creating and signing a transaction. An contract lives inside the Ethereum execution environment, execution specific code when receiving messages or transactions, and managing their own account. Messages are somewhat similar to transactions, but the key differences are that messages are produced by a contract and not an external actor. A message is produced when a contract wants to invoke code on another contract. A message contains:

- The sender of the message (implicit)
- The recipient of the message
- The amount of ether to transfer alongside the message
- An optional data field
- A `STARTGAS` value

A transaction on the other hand refers to the piece of data, signed by an external actor. There are three type of transactions: contract creation transaction, normal transaction with a regular ether transfer and message call transaction to a smart contract. Transactions contain:

- The recipient of the message
- A signature identifying the sender
- The amount of ether to transfer from the sender to the receiver
- An optional data field
- A `STARTGAS` value
- A `GASPRICE` value

The Ethereum blockchain tracks the state of every account, and all state transition function transfers value and information between accounts. Each block contains a copy of the transaction list and the most recent state. Storing the state of every account with every block may seem highly inefficient at first glance, but handled by a special kind of tree known as a ”Patricia tree”. The state is stored in a tree structure, and since after every block, only a small part of the tree needs to be changed. Between two adjacent block, the vast majority of the tree stay the same, and therefore the data can be stored once and then referenced using pointers. A modified merkle-patricia-tree is used to accomplish this.

2.3.4 Consensus model

Mining in Ethereum works the same as in Bitcoin[21]. Miners produce blocks that is only valid if it contains a proof-of-work of a given difficulty. Ethereum uses a different hashing algorithm than Bitcoin called ”Ethash”[22] and the difficulty dynamically adjusts so that on average one block is produced by the network every 15 seconds. Ethash PoW is memory hard, making it basically resistant to highly efficient application-specific integrated circuits(ASIC). The mining rewards are similar to Bitcoin, the successful proof-of-work miner of the winning block receives:

- A static block reward, consisting of five ether
- All of the gas consumed by the execution of all the transaction within the block is compensated for by the senders
- An extra 1/32 of the static block reward per uncle included as part of the block

Since the block time is low relative to Bitcoin’s block time, there is a higher chance that a miner will working on a stale block. A protocol called ”GHOST” (Greedy Heaviest Observed Subtree) was introduced as a way of combating the way that fast block time suffers from a high stale rate[23]. Because blocks take a certain time to propagate through the network, if miner a mines a block and then miner B happens to mine another block before A’s block propagates to B, B’s block will end up wasted and will not contribute to the network security. GHOST solves this issue by including stale blocks in the calculation of which chain is the ”longest”, known as uncles(Figure 2.3.1). Ethereum does more with uncles by rewarding the miner of the uncle that are included in a block with 87.5% of the static block reward. A block can only include uncles up to seven generation to avoid having unlimited uncles.

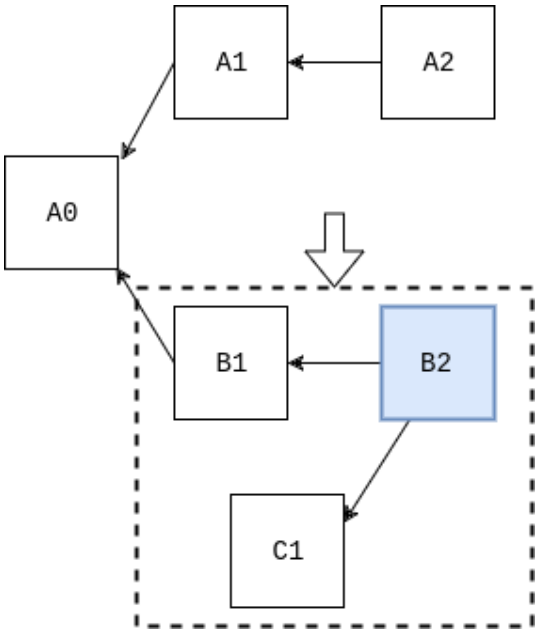

Figure 2.3.1: When B2 is mined, the B branch is selected as the best chain, since C1 is included as an uncle and counts as another block in the branch weight.

2.3.5 Token systems

Ethereum as a platform to distribute decentralized applications is much more powerful in the way that it is not limited to only a cryptocurrency. There are three types of applications on top of Ethereum. The first category is financial applications, proving users a way to manage and entering into contracts using their money. This includes sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and maybe even some classes of full-scale employment contracts. The second category is semi-financial applications, where money is one part of the application but it also involves a non-monetary part to what is being done. For example a self-enforcing bounties for solutions to computational problems. Finally, there are applications such as online voting and decentralized governance that are not financial at all[24].

Token systems are used in many applications ranging from sub-currencies representing assets such as USD or gold to company stock, individual tokens representing smart property, that is, property whose ownership is controlled via the blockchain, secure unforgeable coupons, and even token system with no ties to conventional value at all which is used as point system for incentivization. Token systems are easy to implement in Ethereum since all that a currency, or token system is, is only a database with one operation: subtract X from A and give X units to B. With a condition that A had at least X units before the transaction and the transaction is approved by A. For example a simple hedging contract would look as follows:

1. Wait for party A to input 1000 ether
2. Wait for party B to input 1000 ether
3. Record the USD value of 1000 ether, this value can be retrieved from a ”data feed” such as NASDAQ, let this value be $x

4. After 30 days, allow A or B to ”reactive” the contract in order to send $x worth of ether to A and the rest to B. The new value of x is again retrieved from a data feed.

2.4 Hyperledger Fabric

Hyperledger Fabric is an implementation of a specific type of permissioned blockchain network on which members can track, exchange and interact with digitized assets by using transactions that are controlled by *chaincodes*, which is the piece of code that is installed and instantiated onto the network of Hyperledger Fabric peer nodes, enabling interaction with that network's shared ledger. A permissioned blockchain means that any node is required to maintain a member identity on the network. Even end users must be authorized and authenticated in order to use the network. Participants in the network can interact in a manner that ensures that their transaction and data can be restricted to an identified subset of network participant, known as a *channel*. The members in the channel has the ability to establish a shared ledger containing digitized assets and recorded transaction only available to the members in that channel. There is only one ledger per channel. The ledger is comprised of a blockchain, as well as a state database to maintain current state. The state database represents the latest values for all keys ever included in the blockchain. Nodes execute transaction against the current state to make chaincode interactions extremely efficient. Besides, the blockchain can be stored on nodes file system whereas the state database can reside in memory for fast access. The state database can simply get recovered (or generated if needed) upon node startup, by reading the blockchain from the file system[2].

The Hyperledger Fabric architecture is comprised of the following components: peer nodes, ordering nodes and client applications. These components have identities derived from the certificate authorities. A peer can have two roles; a peer is called a committer when maintaining the ledger by committing transactions, and a peer is called endorser when it is responsible for simulating transactions by executing chaincodes and endorsing the result. Peers are not limited to a single role. A peer may be an endorser for certain types of transaction and just a committer for others. The ordering nodes decides the order of transactions in a block to be committed to the ledger. The ordering service can be implemented as a centralized or decentralized service. There are a few implementations of the ”ordering” function available, e.g. Kafka for crash fault tolerance, or sBFT/PBFT for byzantine fault tolerance. Developers can also implement their own protocol to plug into the service. The work done by peer and ordering nodes are roughly the same kind of work that miners do in the other blockchain architectures.

2.4.1 Transaction flow

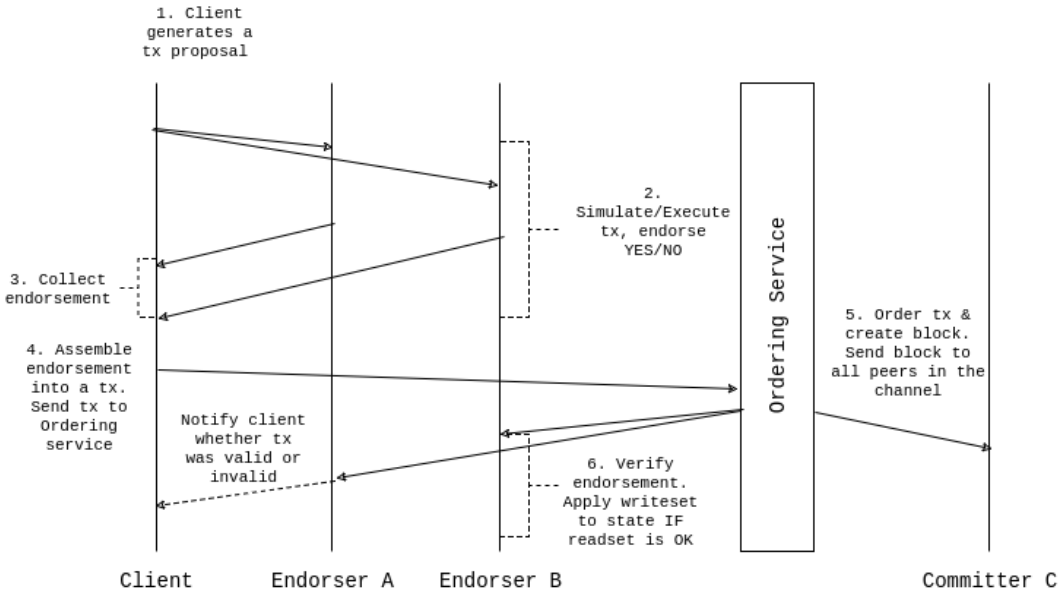

Figure 2.4.1: Illustration of one possible transaction flow.

The basic workflow of a transaction in a Hyperledger Fabric blockchain begins by two or more participant, such as a company or organization that creates and joins a channel. Initially the members in the channel agree on the terms of the chaincode that will govern the transaction as well as the policies governing the channel membership. When a consensus is reached on the proposal to deploy a given chaincode, it is committed to the ledger. Now a end user with the right privileges can propose transaction to endorsers in the channel that executes the chaincode (Figure 2.4.1). The endorsing peer verify the signature of the proposal and determines if the client is authorized to perform the proposed operation. Endorsers use the transaction proposal as input and execute them against the current state database to produce a transaction result. The transaction result includes a response value, read set, and write set. No updates are made to the ledger at this point. The transaction result along with the endorsing peer's signature and a YES/NO endorsement statement are passed back to the client. The client ensures that the results from the endorser are consistent and signed, and send the transaction, comprised of the result, endorsement, and the channel id, to the ordering service. The ordering service does not read the transaction details, it simply orders transactions by channels as First-Come-First-Served basis into a block which is then sent to the peers to be committed to the ledger. The committer nodes validate the transactions within the block to ensure endorsement policy is fulfilled and to ensure no changes has been made to the read set variables in the ledger state. Each transaction in the block are tagged as being valid or invalid and when the peer appends the block to the channel's blockchain; only the write sets of valid transaction are committed to the current state database[25].

2.4.2 Read and write set semantics

When an endorser simulates a transaction, a read-write set is created for the transaction. During simulation, the endorser records what keys are used from the state database to read

their value as well as their committed versions of that value. A list of the keys and their committed version are contained in the read set. The write set contains a list of keys and their new values that the transaction writes. Instead of the new value for a key, a delete marker can be set if the update performed by the transaction is to delete the key.

Following is an illustration of a read-write set prepared by simulation of a transaction.

```
<TxReadWriteSet name"channelId">
    <read-set>
        <read key="K1", version="1">
        <read key="K2", version="1">
    </read-set>
    <write-set>
        <write key="K1", value="V1"
        <write key="K3", value="V2"
        <write key="K4", isDelete="true"
    </write-set>
</TxReadWriteSet>
```

After the transactions are ordered in a block, the committer iterates through the block and checks the validity of each transaction. A transaction is considered valid if the version of each key in the read set matches the version for the same key in the world state, i.e. the current state database. If a transaction passes the validation check, the committer uses the write set to update the world state and the version of the key is changed to reflect the latest version[26].

2.4.3 Endorsement policy

Endorsement refers to the process where endorsing peer nodes execute a transaction and return a YES/NO response to the client application. The endorsement policy defines all the peers that must execute transactions attached to a specific chaincode, as well as the required combination of endorsements. The policy is used to instruct a peer on how to decide whether a transaction is properly endorsed. A policy could require endorsement from a minimum number of endorsing peers, a percentage of endorsing peers, or by all the endorsing peers in the channel[27][28].

3 Method

3.1 Network topology

There are two version of Hyperledger Fabric that can be used for tests: version 0.6 and 1.0. The older version, v0.6, is the only stable release at the moment. The architecture of v0.6 has a few differences compared to v1.0, and most notably is how functions are further separated into more roles. In v0.6, there is only a validating peer and a non-validating peer, whereas in v1.0, there are: endorsing peers, committing peers, and a ordering service. Several significant issues with the architecture of v0.6 was already recognized and are now being addressed in v1.0. For this reason, a decision was made to not use v0.6. Despite being the only stable version which has the tools to monitor and stress test the system that could certainly help in this thesis. Besides, the results the tests would produce would reveal the same kind of issues already recognized and addressed by the developers. Thus, the conclusions would only confirm what has already been addressed to a version that is no longer relevant. For this reason, v1.0 is the sensible version to use in the tests.

Version 1.0 is, however, not finished yet but an alpha version has been released with the functionality needed for this thesis. The problem with v1.0 is the lack of documentation and tools to monitor and stress test the system. With that being said, it was possible to ask the developers and other experienced users how things worked and if you have problems they could help out. Without proper monitoring and stress testing tools, a simple implementation of client that sends request is created, and using the logs to monitor the system.

To start up a fully-functioning transactional network with Hyperledger Fabric there is a couple of fundamental tools that we need to use first.

1. **Prerequisites**
    - Docker - v1.12 or higher
    - Docker Compose - v1.8 or higher
    - Docker Toolbox - Windows users only
    - Go - v1.7 or higher
    - Git Bash - Windows users only
2. **Chaincode**. The chaincode, which is the self-executing logic that encodes the rules for specific types of network transactions. It can be written in Java or Go, but Go is currently more supported than Java, thus, the chaincode is written in Go. The chaincode used in the tests is a simple payment of digital assets from account A to account B. The purpose was to create a simple application found in finance that is not too computational expensive. The reason is that applications that are running slow on the local computer will without a doubt run slow in a Hyperledger Fabric network and there is not much interest in that. What is more interesting to test is how different configuration of the network affect the performance.

3. **Java SDK and Hyperledger Fabric**. A client is created to perform a stress test of the network, which involves: installing chaincode, instantiating accounts, and then send thousands of invoke-request to the network. The invoke-request moves assets from account A to account B, or more specifically, it subtracts X amount from account A and adds X to account B. The Java SDK used in the client is in development as well and the SDK is only compatible with a certain commit-level of Hyperledger Fabric. The commit-level that were used during the test are:

   - Java SDK: `1b6a996f3bb60b06c944cdd4c03ebef8910c7b4f`
   - Hyperledger Fabric: `96b1b901ce2f2f3800a554e4a56c9568954633d5`
   - Hyperledger Fabric-ca: `3b8932ac46e9779b6d1e9aae456816019460bf73`

   Docker images are then created from Fabric and Fabric-ca by issuing `make docker` command in their respective directories. The images are then used to launch peers, orderers, a Certificate Authority(CA), or any other Fabric components in docker containers.

4. **Docker network configuration**. Docker Compose was used to launch our various Fabric component containers, as this is the simplest approach. In the `docker-compose.yaml`-file we can define the services and environmental variable that will be launched with the following command: `docker-compose up`. This was a helpful tool to configure and launch several endorsers with a single command. An example of a `docker-compose.yaml`-file can be seen in the Appendix A.1

5. **Cryptogen-tool**. This tool is used to generate the crypto material. Recall that every entity in the network must be a member and identifiable; this tool generates the certificates and keys for the peers and orderers to digitally sign transactions and prove their membership. The tool can be found in `fabric/common/tools/cryptogen`.

6. **Configtxgen-tool**. The second tool is used to create our first block in the blockchain, i.e. the genesis block for bootstrapping the orderer and a channel configuration artefact. The channel configuration artefact are defined by a `configtx.yaml`-file, which contains the definition of our sample network and present the topology of the network components. The channel configuration used in the test can be seen in the Appendix A.2

7. **Endorsement policy**. Endorsing policy are used to instruct a peer on how to decide whether the transaction is properly endorsed. An example of a endorsing policy is as following: `AND('Org1.member', 'Org2.member')` requires 1 signature from a member i Org1 and a member in Org2. `OR('Org1.member', 'Org2.member')` requires 1 signature from either one of the two members. For critical transactions, e.g. channel configuration transaction, you would probably want endorsement from a administrator with: `AND('Org1.admin', 'Org2.admin')`. The endorsement used in the test are simple since it does not affect performance. A small Go program in Appendix A.3 creates a binary file with the endorsing policy: `AND('Org1.member')`.

4 Result

4.1 Techniques to improve performance and scalability

The different blockchain networks have their own performance and scalability issues. It is mostly due to the implications a public blockchain has compared to a permissioned blockchain. Bitcoin is the public blockchain with the most complicated scalability problem and also the most constrained protocol to change. Ethereum can be implemented as a public blockchain as well as a permissioned blockchain. Both types of Ethereum have different benefits and drawbacks. Hyperledger fabric is only implemented as a permissioned blockchain. What impact does a public blockchain have on performance and scalability compared to a permissioned blockchain? Permissioned blockchains have the advantage to configure the network to allow parallelism whereas public blockchain struggles to partition the network to even make it possible for parallel execution. The problem is really complicated when you consider that no single entity can decide how the network should be partitioned.

The following sections describe techniques to improve performance and scalability in Bitcoin, Ethereum, and Fabric.

Bitcoin

The increasing adoption of cryptocurrencies has raised concerns whether their ability to scale is sufficient. Bitcoin's ability to scale is restricted since its highest transaction throughput has effectively peaked at maximum block size divided by block interval. In recent years the block size has been rapidly filled up towards 1 MB sizes but now in 2017, every generated block has reached the maximum of 1 MB, effectively limiting the transaction throughput to 2-4 transaction/sec. If we look at the current wallet-users which at the time of writing are around 11,8 million user[29], each user can only send around 10 transaction every year[1]. Only ten transaction each year does not allow for any adoption in the financial services and not even near the possibility to allow microtransactions on the blockchain. What if there are 100 million users? This means that each user could only send one transaction each year, which is not sustainable in the economics sector. Without a solution to blockchain congestion problem, transaction on the blockchain will continue to be delayed, and transaction fees will continue to increase, thus, it cannot cover the world's commerce anytime in the near future.

The block size cap was set to 1 MB by Nakamoto in 2008 when the white paper was published. Miners can solve blocks that can be more than 1 MB, but they will be invalid to the rest of the network. This limit can not be raised without a hard fork. This means that nearly all miners need to agree to adopt the new block size, which has been a long debate

[1] 4tps * (365*24*3600) = 10,69

and it even led to splits within the community. The solution to increase the block size is arguably a linear solution, which only creates a breathing room for better scaling solutions. The generated blocks would reach the new block size in no time and each time it is raised through a hardfork

A hardfork is a change to the Bitcoin protocol that makes it less restrictive, non-upgraded miners will not validate blocks created by upgraded miners, and will remain on their own blockchain-fork indefinitely. Softforks implement new rules to the Bitcoin protocol that makes it more restrictive, all blocks considered valid by the newer version are also valid in the old version. The difference between hardfork and softfork is that hardforks are not forward and backward compatible whereas softforks are. To roll out ”hard” changes to the protocol requires every miner, merchant, and user to upgrade or be left behind. Since softfork is backwards compatible, users can update in their own time[30]. Softfork only requires a majority of the miners to upgrade to enforce the new rules. Although the more miners that accept the new rules, the more secure the network is post-fork. Therefore, softforks should be deployed when 95% of the miners have signaled their support.

There are two camps of the Bitcoin community with different opinion on how to fix Bitcoin's scalability issues. On one side there is a team of developers, known as Bitcoin core that proposed a feature via a softfork aiming to optimize the blocksize. This feature is included in a concept called Segregated Witness, or SegWit, which aims to remove data related to signatures from Bitcoin transaction making them smaller in size. This in turn makes the block even smaller meaning more transaction can be included in the block. SegWit also addresses other issues as transaction malleability and opens up for second layer solutions like the Lightning Network. It requires 95% of miners to signal support for SegWit and for full nodes to upgrade to the latest version to activate this feature[31]. However SegWit has met a lot of resistance from miners who have raised concerns about it's technological and economic complexity in hope of finding a better solution.

A solution for micropayments in Bitcoin is to decide whether all transactions have to be recorded on the blockchain. If only a handful of participants care about an everyday recurring transaction, it not necessary for all other nodes in the network to know about that transaction. Instead, we create micropayment channels between two parties and let them send as many transaction they want. Only a single transaction netting out the total balance between the two parties needs to be recorded on the blockchain. This allows Bitcoin to scale to millions or even billions of transaction per day. It also allows for a trustless channel since one of the parties can always guarantee their current balance on the blockchain if the other party does not cooperate[32].

On the other side are those who wanted an increase in block size, with further debate over what the ideal size was. Bitcoin Unlimited provides a voice for all stakeholders in the bitcoin ecosystem. Every node operator or miner can choose their own blocksize. This approach allows the miners to vote with their hash power on what the block size should be and let them reach a consensus based on free-market economics[33][34]. Although, implementing this feature would involve a hardfork which requires everyone to agree with and this is almost impossible.

Ethereum

Probably the biggest scalability issues with Ethereum are that every node has to process all transactions and has to store the entire state of every account balance, contract code and storage, etc. Although this provides a large amount of security, but greatly limits scalability to the point that a blockchain cannot process more transactions than a single node. One would think that a network with thousands of nodes should be able to have more throughput than a single node, but this is not the case in Ethereum or in public blockchain networks in general.

A possible solution for this problem is to create a new mechanism where only a small subset of nodes has to verify a subset of transactions. As long as there are sufficient many nodes verifying each transaction, the system will still be secure, but also allow for the system to process transactions in parallel. This techniques is called *sharding*. The basic idea behind sharding is by dividing the global state of accounts, both external and contract accounts, in smaller chunks known as a *shard*. In simpler forms of sharding, each shard also has its own transaction history, and the effects of transaction in some shard K are limited to the state of shard K. However, transactions across two shards can be achieved with a ”debit” and ”credit” kind of transactions. For example a transfer of money, where money is moved from shard K to shard L by first creating a ”debit” transaction that destroys coins in shard K, and then creating a ”credit” transaction that creates coin in shard L, pointing to a receipt created by the debit transaction as proof that the credit is legitimate. In more complex forms of sharding, transactions may in some cases affect other shards as well and may also synchronously ask for data from the state of multiple shards. Each shard gets its own set of validators, and these validators will not need to validate all shards[35].

Hyperledger Fabric

The performance and scalability in Fabric are completely different from both Bitcoin and Ethereum due to being a permissioned blockchain. This allows Fabric to have different types of nodes with their own responsibilities, and allows them to configure a network of nodes to scale independent of each other. There is no parallel relationship between the number of peers in a network and the number of orderers. It is possible to add endorsers and committers without having to add corresponding orderers. The system scale better than if these functions were executed by the same nodes. This becomes clear when you consider that chaincodes can be installed on disjointed endorsers, which introduces partitioning of chaincodes between endorsers and allows parallel chaincode execution. Furthermore, endorsers may have to execute heavy computations, but since endorsers are separated from the ordering service it does not affect it’s execution time. Developers are therefore free to write more complex applications that are costly to execute without disrupting the ordering service or any other application on the network.

Fabric has a concurrency control where transactions are executed in parallel by the endorsers to increase throughput. Since endorsers do not have to worry about the ordering of transaction and double spending, because they are handled by the ordering service and the committers, they can push through as many transaction per second as possible. Comparing with Ethereum, where the nodes must execute transactions sequentially to prevent double spending, will not benefit much by using powerful hardware while endorsers could utilize the hardware to full potential. This means that we got hardware scaling and it make sense

for enterprises to invest in powerful computers to run as endorsers.

Channels are a clever way to split one big blockchain into many private blockchains, allowing for data isolation and confidentiality. Each channel has a corresponding ledger which is shared across the peers in the channel. The channel-specific ledger contains the blockchain and a state database with current state data. The peers in the channel maintain their own copy of the ledger and syncs ledger state across all peers using a gossip data dissemination protocol. It is the gossip protocol that connects the peers on a channel to spread ledger data and manages peer discovery and channel membership. Combine the protocol with channels and we got a scalable network, where peers only need to communicate with other peers in the same channel. Adding more peers to a channel should affect the performance on just that channel, other channels will not be affected considering how separated channels are from each others. Although you could take advantage of a more coarse grained endorsement policy to improve performance on the channel when adding peers. Lets say you need N endorsement out of M to have a properly endorsed transaction and the endorsement can be signatures from any of the members of organisation: Org1. The transaction could be load balanced to any of M members and since the clients does not know the identities of the endorsers we get elasticity, i.e. the number of members M can grow/shrink on demand[36][37][38].

4.2 Performance analysis and the three-way trade-off

Proof-of-Work way of extending the blockchain heavily and negatively impacts system scalability and overall throughput. Bitcoin has been forked a number of times in order to tune PoW parameters, i.e. the block generation time and the hash function. The most popular fork, or altcoin, is known as *Litecoin* which has a 2,5 minutes block time instead of a 10 minutes block time[39]. However, in terms of transaction speed, a protocol with shorter block time needs more confirmations for the same level of security as a protocol with longer block time. A confirmation is not a guarantee of authenticity. A confirmation is indirectly determined by difficulty, with difficulty roughly determines how many hashes are required to solve a block. If a block time of 2,5 minutes are used instead of 10 minutes, the number of hashes required is roughly 1/4, which essentially means that the ”value” of a confirmation is worth about 1/4 of a confirmation on 10 minutes block time. Although, one confirmation time will be faster and it is usually adequate for most transaction, where the potential loss due to double spending is small. Merchant and exchanges, who arguably bear more risk from double spending would still want the same level of security as six confirmations on a ten minutes block time. Generally shorter block time means a higher stale rate which in turn requires a higher number of confirmation to match Bitcoin’s security. A higher stale rate also means that more work is wasted. The ten minutes block time is simply a compromise.

Block size is also one of the main performance-related parameter of a PoW blockchain such as Bitcoin. The 1 MB block size is a compromise as well. Increasing the block size with the goal to increase throughput comes as the cost of increasing the latency, because of longer propagation delays of larger blocks across the network. These longer delays, in turn, have negative implications on Bitcoin security: longer delays may increase the risk of chain forking and therefore increases the possibility for dishonest miners to mount a double-spend attack. With increased block size comes a larger blockchain, which means the blockchain could get too large in size that not everyone can run a full node. This means that

only large scale server can run a full node, leaving the average user behind, and therefore resulting in less decentralization. The main concern with full node centralization is trust: if there is only a few entities capable of running full nodes, then those entities could conspire to allow invalid transactions for their own gain, and there will be no way for other users to know without processing the block for themselves. Furthermore, larger blocks cause miner-centralization as well since it takes more time to download larger blocks. Thus users on lower bandwidth can not compete with users on higher bandwidth and only a small collection of businesses can afford the resources.

Ethereum has mitigated most of the security loss with faster block time by using a modified version of GHOST protocol, but the blocks still have to propagate across the network which is a relatively slow process. The block size needs to be smaller in order to propagate faster and that is why Ethereum can only process about 15 tps even though the block time is as low as 15 seconds. The bottom line is that public blockchain networks are not efficient. It is the price we pay for decentralization and it does not matter which network we are talking about, neither of them does not even come close to providing the kind of throughput that would truly be needed for global adoption. It comes down to every single node has to bear the full transaction load of the entire network. These two public blockchains do not scale well.

Ethereum is, on the other hand, in far worse state than Bitcoin since all the Bitcoin network need to do is to handle simplistic transactions. Ethereum is supposed to do much more than that. It is Turing complete and the network needs to handle arbitrary processing tasks and store potentially immense amount of data. That is why Ethereum is working on the most promising scaling solutions compared to Bitcoin. To be able to scale and to provide more throughput we need a solution for two bottlenecks: transaction processing and state storage. Channel-based strategies like lightning network, Raiden, etc., can scale transaction processing by a constant factor but does not scale state storage. The most promising solution is on-chain scaling via sharding and complementary off-chain scaling via channels, but it will take some time before these solutions are implemented in production. The reason is that blockchains gets their robust decentralized trust from the way they operates. They are secure for the same reason they do not scale, changing something to gain scalability while maintaining high security is a difficult task.

When comparing these two public blockchain network with a permission blockchain such as Hyperledger Fabric—we can see that sacrificing some of decentralization greatly improves scalability and performance. Sacrificing some decentralization means we need more trust on the validating nodes and the authority and trust has to come from outside the network. By having a governing authority that provides inherent level of trust between participants, enables design decisions like sharding and channels to be implemented without much hassle. And since trust is already achieved from outside the network, we do not need much computational power to support that trust. This allows us to use different consensus mechanism, such as BFT protocols, and almost all of the issues public blockchains have, disappears. Furthermore, the governing body can ensure data access to participants in the channel and only allow them to view sensitive transaction data. Thus, Hyperledger Fabric has the features required for modern enterprise security standards, that are difficult to achieve in public blockchains.

4.3 Application requirements

Because the term ”blockchain” is not clearly defined it can be difficult to know whether an application can run on a blockchain or if it is just better to run with a regular database. In most cases a regular database is more appropriate way to go, and using a blockchain would just be considered as over-engineering. It could make sense to use a blockchain if robustness and disintermediation is important, otherwise, there is nothing a blockchain can do that a regular database cannot. Additionally, smart contract are only computer code running on every node in the network, and it is similar as stored procedures in DBMS.

If the application does make sense to run on a blockchain there are a few requirements to think about during development if the application will run on Ethereum or Hyperledger Fabric.

Since every node in the Ethereum network will execute the same code, a gas is introduced to pay for the resources needed, which was described in more detail in the Ethereum chapter. The gas is used to pay for each computational step, memory usage, storage, and network usage. This means that heavy computations are really expensive as well as sending huge data packages. Therefore, an application should be developed as simple as a ordinary calculator but will have great security. The gas prevents attackers from deploying applications with infinite loops.

Another important requirement is that the smart contract must be deterministic. We cannot have each node returning different result, then the network will not reach a consensus. It is not possible for smart contracts to make API-calls to the Internet since the data fetch from different sources are considered dynamic data and such sources would interfere with the consensus process of Ethereum. That is why smart contracts can only fetch data within their own blockchain. The method to fetch data from outside of the blockchain can be achieved by *oracles*. An oracle is a third party which provides a smart contract with specific data from the outside world[40]. However, this means that the smart contract is no longer decentralized since the data comes from a single entity that could manipulate the responses. This risk can be mitigated by allowing multiple oracles to have control over a M-of-N multi signature contract, that requires at least M signatures before the transaction is accepted. Its arguably more decentralized this way and other contracts can rely on the external information since it is much easier for N nodes to reach consensus on the result of an HTTP request than an entire blockchain.

We are also not sure on what hardware the nodes are running on. It is important for anyone to join the network without requiring a supercomputer to keep the network as decentralized as possible. This problem does not really exist in Hyperledger Fabric since the endorsers that executes the contracts are known, and they are only a couple of nodes with that responsibility compared to Ethereum that has at least 5000 nodes. Furthermore, since the endorsers are chosen we can demand higher performance hardware. This would allow for more demanding applications.

Unlike Ethereum, Fabric does not have a way to control the amount of resource used like gas does. Fabric allows application to use as much memory, storage, bandwidth, and can execute as many computational step as they want. They rely on the fact that everyone has a membership and can therefore be identifiable. If an application misuse the resources, appropriate action will be taken on the developers.

4.4 Blockchain or traditional centralized solutions?

There is also, however, a further point to be considered is whether using blockchain technology in a application add more value to the product than using a regular relational database as today's traditional centralized solutions have been using for decades. We have already explored the limitations of different blockchain networks compared with each other in previous sections, but what are the limitations of blockchains compared with traditional centralized solutions?

The answer to this question is partially answered already in previous sections. We have concluded that every single node has to bear the full transaction load of the entire network. Moreover, the process of validating each transaction is not only applying the changes to the ledger like a regular database, but the node also carries three additional burdens:

1. **Signature verification**. Every transaction signature must be verified and verification of these signatures are computationally complex. A regular centralized database does not need to individually verify every transaction once a connection has been established.
2. **Consensus mechanism**. A blockchain has to be able to create trust in a naturally distrustful situation and to ensure that nodes in the network reach consensus. It requires immense amount computational power to support that trust and we have seen that the rather long time it takes to reach consensus is necessary to have a highly secure system. While a centralized database does not need to create trust in the system since the database is owned and controlled by the same organisation.
3. **Redundancy**. The fact that every transaction must be processed independently by every node in the network for the same end result. Whereas centralized databases process transactions once.

Today's blockchains will therefore always be slower than centralized solutions. It is not because blockchain is still new and unoptimized, but it is result of the nature of blockchains themselves.

4.5 Performance constraints in Hyperledger Fabric

Block size scaling

The configuration of the Fabric can be done in many areas of the network. Enterprises can decide for themselves the composition of peers by deciding how many endorsers, committers and orderers that suits their use case. Furthermore, there is one parameter, `MaxMessageCount`, that can be optimized to maximize throughput. This parameter tells the orderer how many transaction that can be included inside a batch which is then sent to committers to form the next block.

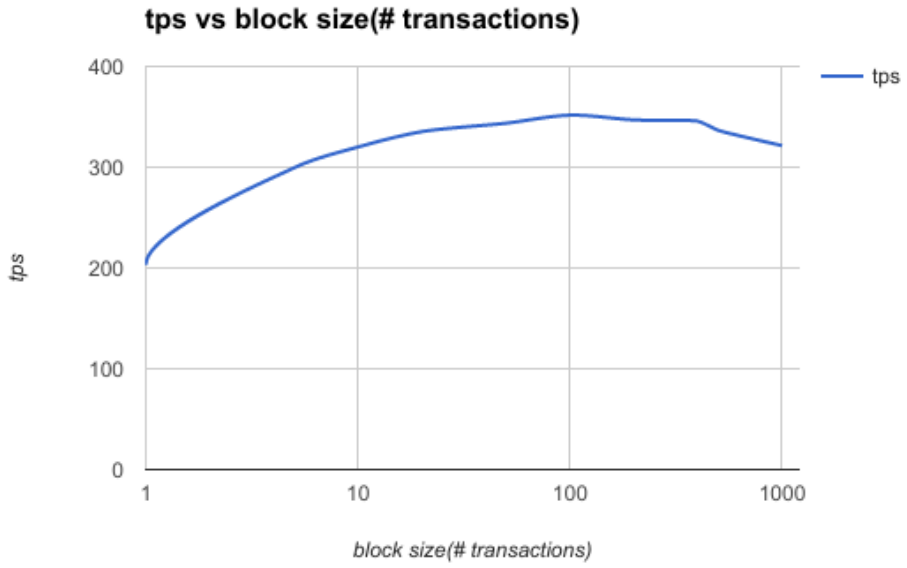

Figure 4.5.1: Transaction per second(tps) as a function of block size(# transactions)

As we can see in Figure 4.5.1, only a couple of transactions per block do have a negative impact on throughput. Although, the throughput quickly increases at 10 transaction per block then the performance increase starts to diminish. The maximum throughput of 350 tps lands around 100 transactions per block with this network configuration.

Endorser scaling

Here we mainly focus on endorser scaling on the same channel. From Section 4.1 we concluded that adding endorsers on different channels would not have any impact on the performance of a channel. Therefore, we limit the test to a single channel.

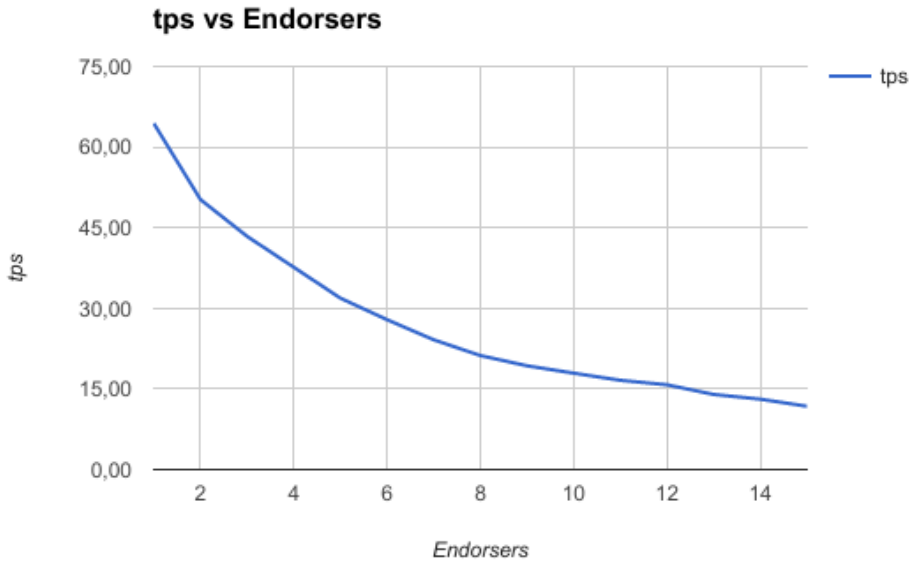

Figure 4.5.2: Transaction per second(tps) decreases with larger networks

The test is configured with a single client that uses 1 thread to send 1000 transactions to varied amount of endorsers and record the time taken for all transaction to be included in a block (Figure 4.5.2). The most significant drop in throughput; from one endorser to two endorsers; is the most interesting part.

5 Discussion

The rather low throughput we got from the test does not reflect the performance of Hyperledger Fabric in reality. Fabric could handle thousands of transactions per second if powerful machines are used to run as endorsers and multiple clients sends as many transactions they can. The fact that this test use the same machine to run as an endorser and as a client means that they have to share resources. We can only achieve as much tps as one computer can handle. It is always tough to test performance of a distributed system, such as this one, with limited resources. To truly reach a maximum throughput, one would have to run tests on a production-ready environment with several powerful computers connected with each other in close proximity. Only then can you find out how powerful this type of system can be, and using just one computer only proves that the computer is the bottleneck, while the purpose was to find out where the bottlenecks are in the system itself.

What you can do is to divide the resources evenly between the peers. Each peer is limited to only one core, and if we control the CPU usage and not push the peers to use 100% of their core, then we can see a trend to find possible bottlenecks. The performance would be lower, naturally, but this isolates peers more from each other and it almost emulates peers running on completely different machines. However, there is no denying that some peers will be affected more than others by background noise, i.e. processes from other applications running in the background, but there is not much you can do about it.

The result also shows that more endorsers in the network lowers the throughput. Remember that endorsers execute chaincode independent from one another, which means that the messaging from the gossip protocol has a significant impact on performance and scalability. We get the most performance out of Fabric with one endorser, but this to be expected as this configuration of the network is as close to a centralized solution as possible using Fabric. The gossip protocol has not taken effect yet since there is not another endorsers to exchange messages with. The performance drop rather quickly when you add the second endorser and the effects of the gossip protocol kicks in. As more endorsers are added to the network, each endorser has to exchange messages between every other endorsers across the channel which is why we see a continuous decrease in throughput. This intensive network communication would arguably involve as many as $O(n^2)$ messages per endorser, where n is the number of endorsers in the channel. Thus, the scalability of the number of endorsers are limited to only a couple endorsers per channel, but it is not well explored yet. Once again, one would have to test on a standard production environment with powerful endorsers running in close proximity to find out how the number of endorsers scale. Furthermore, BFT-based blockchain offers good performance for small number of nodes, whereas PoW-based blockchain offers good node scalability with poor performance. Given seemingly inherent trade-off between the number of nodes and performance, enterprises should strive to use a small number of endorsers for their use cases.

There is also, however, a further point to be considered. These tests does not take network latency and bandwidth into account. Bandwidth could certainly have an impact on the

block size, where larger blocks would take more time to send compared to smaller blocks. The network latency would impact the messaging which is constantly going on across the endorsers in the channel. That is why enterprises or a consortium of financial institutions should put their computers that runs as peers in close proximity to minimize the latency as much as possible.

6 Conclusion

This research examined the differences between a public blockchain with a permissioned blockchain in regards to performance and scalability and how viable Hyperledger Fabric is to use within the finance industry compared to traditional centralized solutions. The results have shown that sacrificing decentralization by creating authority and trust outside of the network greatly improves performance and scalability. A permissioned blockchain does not require computational power to support that trust compared to public blockchains, which in turn eliminates the throughput issues.

Hyperledger Fabric is certainly much faster and scalable than both Bitcoin and Ethereum, and it can ensure data access to allow only the participant in a party to a transaction can see sensitive details. For many enterprise and financial institutes use cases, Hyperledger Fabric can meet the business requirements that are impossible to meet with a public blockchain. Although the question still stands if Hyperledger Fabric is fast and scalable enough to replace the the centralized systems used today. The results have shown that Hyperledger Fabric can put up with much higher throughput than Bitcoin and Ethereum but to truly know the performance and scalability of Hyperledger Fabric one would have to run tests in a network of powerful computers.

References

[1] Ethereum Foundation. Ethereum webpage. https://www.ethereum.org/. [Online; accessed 2017].

[2] Hyperledger webpage. https://www.hyperledger.org/. [Online; accessed 2017].

[3] Chaincode. http://hyperledger-fabric.readthedocs.io/en/latest/smartcontract.html?highlight=chaincode. [Online; accessed 2017].

[4] Blockchain (database). https://en.wikipedia.org/wiki/Blockchain_(database), 2017. [Online; accessed 2017].

[5] Vitalik Buterin. Merkling in Bitcoin. https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/, 2015. [Online; accessed 2017].

[6] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008. [Online; accessed 2017].

[7] Stale blocks. https://bitcoin.org/en/glossary/stale-block. [Online; accessed 2017].

[8] Orphan blocks. https://bitcoin.org/en/glossary/orphan-block. [Online; accessed 2017].

[9] Bitcoinwiki. Block. https://en.bitcoin.it/wiki/Block. [Online; accessed 2017].

[10] Learncryptography. 51% attack. https://learncryptography.com/cryptocurrency/51-attack. [Online; accessed 2017].

[11] Bitcoinwiki. Confirmation. https://en.bitcoin.it/wiki/Confirmation. [Online; accessed 2017].

[12] Bitcoin network stats. http://bitcoincharts.com/bitcoin/. [Online; accessed 2017].

[13] Top 10 sites for November 2016. https://www.top500.org/lists/2016/11/. [Online; accessed 2017].

[14] Application-specific Integrated Circuit. https://en.wikipedia.org/wiki/Application-specific_integrated_circuit. [Online; accessed 2017].

[15] Mining Hardware Comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison. [Online; accessed 2017].

[16] Hashrate Distribution. https://blockchain.info/sv/pools/. [Online; accessed 2017].

[17] Ethereum Foundation. Ethereum White-Paper. https://github.com/ethereum/wiki/wiki/White-Paper. [Online; accessed 2017].

[18] Ethereum wiki. https://en.wikipedia.org/wiki/Ethereum. [Online; accessed 2017].

[19] Solidity programming language. http://solidity.readthedocs.io/en/latest/index.html. [Online; accessed 2017].

[20] What is Ether. https://www.ethereum.org/ether. [Online; accessed 2017].

[21] Mining in Ethereum. https://github.com/ethereum/wiki/wiki/Mining. [Online; accessed 2017].

[22] Ethash. https://github.com/ethereum/wiki/wiki/Ethash. [Online; accessed 2017].

[23] Ethereum Modified GHOST implementation. https://github.com/ethereum/wiki/wiki/White-Paper#modified-ghost-implementation. [Online; accessed 2017].

[24] Ethereum applications. https://github.com/ethereum/wiki/wiki/White-Paper#applications. [Online; accessed 2017].

[25] Hyperledger Fabric Transaction flow. http://hyperledger-fabric.readthedocs.io/en/latest/txflow.html. [Online; accessed 2017].

[26] Hyperledger Fabric Read-Write set semantics. http://hyperledger-fabric.readthedocs.io/en/latest/readwrite.html. [Online; accessed 2017].

[27] Hyperledger Fabric Glossary. http://hyperledger-fabric.readthedocs.io/en/latest/glossary.html. [Online; accessed 2017].

[28] Hyperledger Fabric Endorsing policy. http://hyperledger-fabric.readthedocs.io/en/latest/endorsement-policies.html. [Online; accessed 2017].

[29] Bitcoin wallet users. https://blockchain.info/charts/my-wallet-n-users. [Online; accessed 2017].

[30] Blockchain Rule Update Process. https://gist.github.com/gavinandresen/2355445. [Online; accessed 2017].

[31] Segregated Witness Costs and Risks. https://bitcoincore.org/en/2016/10/28/segwit-costs/#introduction. [Online; accessed 2017].

[32] The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf. [Online; accessed 2017].

[33] Bitcoin Unlimited Wiki. https://en.wikipedia.org/wiki/Bitcoin_Unlimited. [Online; accessed 2017].

[34] What is Bitcoin Unlimited. https://www.cryptocompare.com/coins/guides/what-is-bitcoin-unlimited/. [Online; accessed 2017].

[35] On Sharding Blockchains. https://github.com/ethereum/wiki/wiki/Sharding-FAQ. [Online; accessed 2017].

[36] Hyperledger Fabric Channels. http://hyperledger-fabric.readthedocs.io/en/latest/channels.html. [Online; accessed 2017].

[37] Hyperledger Fabric Ledger. http://hyperledger-fabric.readthedocs.io/en/latest/ledger.html. [Online; accessed 2017].

[38] Hyperledger Fabric Gossip Data Dissemination Protocol. http://hyperledger-fabric.readthedocs.io/en/latest/gossip.html. [Online; accessed 2017].

[39] What is the difference between Litecoin and Bitcoin? http://www.coindesk.com/information/comparing-litecoin-bitcoin/. [Online; accessed 2017].

[40] Ethereum and Oracles. https://blog.ethereum.org/2014/07/22/ethereum-and-oracles/. [Online; accessed 2017].

A Appendix

A.1 Docker-compose.yaml

```
version: '2'

services:
  ca:
    container_name: ca
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/var/hyperledger/fabric-ca-server
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start --ca.certfile /var/hyperledger/fabric
       -ca-server-config/peerOrg1-cert.pem --ca.keyfile
    /var/hyperledger/fabric-ca-server-config/267
       da8fb0b04f06e62b0d97ed82fb692c668640227947192f26955390ce72b53_sk -b admin
       :adminpw -d'
    volumes:
      - ./crypto-config/peerOrganizations/peerOrg1/ca/:/var/hyperledger/fabric-
        ca-server-config

  orderer:
    container_name: orderer
    image: hyperledger/fabric-orderer
    environment:
      - ORDERER_GENERAL_LOGLEVEL=INFO
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/msp/orderer
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/orderer
    command: orderer
    volumes:
    - ./channel:/var/hyperledger/configtx
    - ./crypto-config/ordererOrganizations/ordererOrg1/orderers/
       ordererOrg1orderer1/:/var/hyperledger/msp/orderer
    ports:
      - 7050:7050

  peer1:
    container_name: peer1
    extends:
        file: peer-base/peer-base.yaml
        service: peer-base
    cpuset: "1" # specify which core to use
    environment:
```

```
      - CORE_PEER_ID=peer1
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer1:7051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1:7051
    volumes:
        - /var/run/:/host/var/run/
        - ./crypto-config/peerOrganizations/peerOrg1/peers/peerOrg1Peer1/:/var/
          hyperledger/msp/peer
    ports:
      - 7051:7051
      - 7053:7053
    depends_on:
      - orderer
```

A.2 Configtx.yaml

```
---
################################################################################
#
#   Profile
#
#   - Different configuration profiles may be encoded here to be specified
#   as parameters to the configtxgen tool
#
################################################################################
Profiles:
    # SampleSingleMSPSolo defines a configuration which uses the Solo orderer,
    # and contains a single MSP definition (the MSP sampleconfig).
    SampleSingleMSPSolo:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1

################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:
    # SampleOrg defines an MSP using the sampleconfig. It should never be used
    # in production but may be used as a template for other definitions.
    - &OrdererOrg
        # DefaultOrg defines the organization which is used in the sampleconfig
        # of the fabric.git development environment.
        Name: OrdererMSP

        # ID to load the MSP definition as.
```

```
    ID: OrdererMSP

    # MSPDir is the filesystem path which contains the MSP configuration.
    MSPDir: /home/mattiasscherer/Documents/fabricbenchmark/myorg/crypto-
        config/ordererOrganizations/ordererOrg1/orderers/ordererOrg1orderer1

    # BCCSP: Select which crypto implementation or library to use for the
    # blockchain crypto service provider.
    BCCSP:
        Default: SW
        SW:
            # TODO: The default Hash and Security level needs refactoring to
                be
            # fully configurable. Changing these defaults requires
                coordination
            # SHA2 is hardcoded in several places, not only BCCSP
            Hash: SHA2
            Security: 256
            # Location of key store. If this is unset, a location will
            # be chosen using: 'MSPDir'/keystore
            FileKeyStore:
                KeyStore:

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

    # ID to load the MSP definition as
    ID: Org1MSP

    # MSPDir is the filesystem path which contains the MSP configuration
    ##############################################################################

    # FIXME: this path needs to be fixed to point to the actual location of
        #
    # the project 'fabric-sdk-node' in the file system
        #
    ##############################################################################

    MSPDir: /home/mattiasscherer/Documents/fabricbenchmark/myorg/crypto-
        config/peerOrganizations/peerOrg1/msp

    # BCCSP (Blockchain crypto provider): Select which crypto implementation
        or
    # library to use
    BCCSP:
        Default: SW
        SW:
            Hash: SHA2
            Security: 256
            # Location of Key Store. If this is unset, a location will
            # be chosen using 'MSPDir'/keystore
            FileKeyStore:
```

```
                    KeyStore:

        AnchorPeers:
            # AnchorPeers defines the location of peers which can be used
            # for cross org gossip communication.  Note, this value is only
            # encoded in the genesis block in the Application section context
            - Host: peer0
              Port: 7051


################################################################################
#
#   SECTION: Orderer
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for orderer related parameters
#
################################################################################
Orderer: &OrdererDefaults

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

    Addresses:
        - orderer:7050

    # Batch Timeout: The amount of time to wait before creating a batch.
    BatchTimeout: 10s

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

        # Max Message Count: The maximum number of messages to permit in a
        # batch.
        MaxMessageCount: 100

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

        # Preferred Max Bytes: The preferred maximum number of bytes allowed for
        # the serialized messages in a batch. A message larger than the
        # preferred max bytes will result in a batch larger than preferred max
        # bytes.
        PreferredMaxBytes: 2048 KB

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

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

```
    # the orderer side of the network.
    Organizations:
################################################################################
#
#   SECTION: Application
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for application related parameters
#
################################################################################
Application: &ApplicationDefaults

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

A.3 Endorsing policy program

```
package main

import (
        "fmt"
        "io/ioutil"

        "github.com/hyperledger/fabric/common/cauthdsl"
        putils "github.com/hyperledger/fabric/protos/utils"
)

const policyString = "AND('Org1MSP.member')"

func main() {
        p, err := cauthdsl.FromString(policyString)
        if err != nil {
                fmt.Printf("Invalid policy %s\n", policyString)
        } else {
                policyMarshalled := putils.MarshalOrPanic(p)
                err = ioutil.WriteFile("./policyBits", policyMarshalled, 0777)
                if err != nil {
                        fmt.Printf("Error writing to file: %s", err)
                }
        }
}
```