Masaryk University
Faculty of Informatics

Blockchain open-source software comparison

Master's Thesis

Bc. Dávid Urbančok

Brno, Fall 2019

Replace this page with a copy of the official signed thesis assignment and a copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Dávid Urbančok

Advisor: RNDr. Michal Batko, Ph.D.

Acknowledgements

I would like to thank my supervisor, RNDr. Michal Batko, Ph.D., who has helped me a lot with various invaluable inputs into this thesis.

I would also like to extend my gratitude to Silvia Sýkorová for a never-ending support and encouragement while writing this thesis.

And lastly, I would like to express my thankfulness to my friend Norbert Fábián who was willing to lend me his laptop for testing when Murphy's law took my computer away two weeks before the deadline.

Abstract

Blockchains are rapidly developing technologies that offer a lot of interesting ideas and features. They are the technology powering cryptocurrencies, virtual currencies, that gained a lot of attention causing the Cryptocurrency bubble.

This diploma thesis is aimed at describing and comparing four open-source blockchain technologies - Ethereum and Hyperledger's blockchains - Fabric, Besu, and Iroha.

The goal of this thesis is to provide a comprehensive comparison of the functionality and performance of the selected blockchains, and evaluate the advantages and disadvantages of each blockchain.

The thesis will describe what a blockchain technology is, on what principles is it built, what are the data structures used and the most common attack aimed at blockchains.

Next, the thesis outlines the inner-workings of each blockchain - how are they implemented from a high-level point-of-view, how are transactions executed and how is the consensus achieved.

Later, the thesis will run test cases for performance benchmarking and collect statistical data for analysis and comparison.

Lastly, the thesis will compare the blockchains and discuss the advantages and disadvantages of each tested blockchain.

Keywords

blockchain, consensus, performance, transactions per second, latency, Ethereum, Fabric, Besu, Iroha

Contents

List of Tables

List of Figures

1 Introduction

When *Bitcoin*, the most popular cryptocurrency, became famous with the general public, people of all sorts started looking into investments in cryptocurrencies.

With the emerging popularity of Bitcoin, the technology that powers cryptocurrencies, *blockchain*, also gained a lot of attention. Since then, a number of *blockchain* technologies emerged - some proprietary, some open-source. Some of the *blockchains* backed by a cryptocurrency, others not.

Blockchain technology promises a widespread usage in different sectors of the market and industry thanks to its characteristics.

The aim of this thesis is to describe and compare the open-source implementations of *blockchain*. The comparison is done from two different points of view: *functionality* and *performance*.

The second chapter begins by describing what a blockchain is, what data structures are present in a blockchain, and what are the pillars on which blockchains are built. This chapter also provides a brief description of the use cases of blockchain, the fundamentals of *mining* and *consensus mechanisms*. The second chapter further discusses the most common attacks on a blockchain, what blockchain types exist and what metrics are usually present when performance testing a blockchain network.

The third chapter describes *Ethereum*, a blockchain backed by *Ether*, the second most popular cryptocurrency at this time. This chapter also provides an insight to the states of an Ethereum network, the gas that is used to prevent attacks by introducing infinite loops into transactions and a reward for miners for expending computational power. Furthermore, the third chapter discusses the transaction execution on an Ethereum network.

In the fourth chapter, the *Hyperledger Fabric* blockchain is presented. This chapter describes the channels and the types of nodes on a *Fabric* network. Moreover, the transaction flow, the ordering and Zookeeper services are also discussed. Lastly, the *Kafka* ordering service's purpose in a *Fabric* network is described.

The fifth chapter introduces the *Hyperledger Besu* blockchain. The chapter also describes the variety of the consensus protocols Besu can use, how is transaction privacy achieved, and how granular permissioning works in Besu.

The sixth chapter briefly discusses the *Hyperledger Iroha* blockchain and its unique Yet another consensus algorithm. This chapter also describes the transaction execution in Iroha.

The seventh chapter presents the tool used to set up and test the blockchains. This chapter also contains a description of the test cases for performance benchmarking the selected blockchains. This chapter also contains the benchmark results for each test case.

The eight chapter compares the selected blockchains head-to-head. This means comparing the consensus mechanisms, transactions, their verification, and various performance indicators. This chapter also contains a brief discussion of each blockchain's advantages and disadvantages.

Appendix A contains the prerequisites and commands to run the performance benchmarks on a local workstation while Appendix B contains the performed benchmarks' full results for each test case.

2 Blockchain

A **blockchain** (or *block chain*) is a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties. Each transaction in the public ledger is verified by consensus of a majority of the participants in the system. Once entered, information can never be erased. The blockchain contains a certain and verifiable record of every single transaction ever made[1].

Alternative definition:

A blockchain is a **cryptographically secure transactional singleton machine** with shared-state[2].

- **Cryptographically secure** - The creation of digital currency is secured by complex mathematical algorithms.
- **Transactional singleton machine** - There is a single canonical instance of the machine responsible for all the transactions being created in the system.
- **With shared-state** - The state stored on this machine is shared and open to everyone.

The distributed database of records (blocks) is linked using *cryptography*. Each block contains a hash pointer of the previous block, a timestamp, and transaction data.

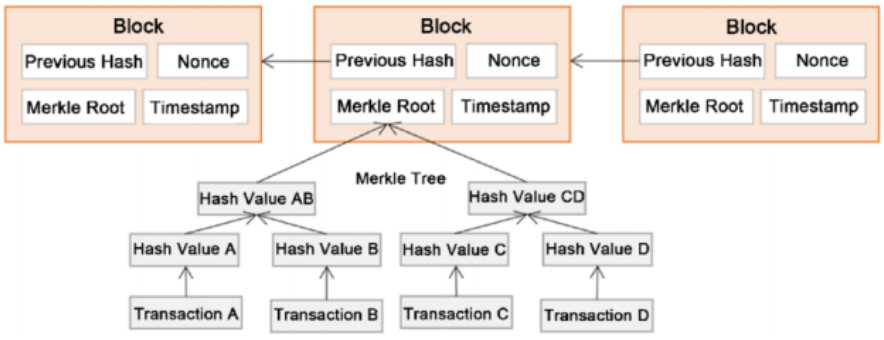

Figure 2.1: Blockchain structure

2.1 Cryptographic hash functions

A hash function is a function with the following three properties:

- Its input can be any string of any size.
- It produces a fixed-sized output.
- It is efficiently computable.

For a hash function to be cryptographically secure, it is required that it has the following three additional properties: (1) collision resistance, (2) hiding, (3) puzzle friendliness.

2.1.1 Collision resistance

A collision occurs when two distinct inputs produce the same output. A hash function is collision resistant if nobody can find a collision.

Collisions exist for any hash function and this can be proven by a simple **counting argument**. The input space to the hash function contains all strings of all lengths, yet the output space contains only strings of a specific fixed length. Because the input space is infinitely larger than the output space, there must be strings that map to the same output string.

2.1.2 Hiding

The hiding property asserts that if given the output of the hash function $y = H(x)$, there is no feasible way to figure out what the input of function H, x, was.

2.1.3 Puzzle friendliness

If someone wants to target the hash function to have some particular output value y, and if part of the input has been chosen in a suitably randomized way, then it is very difficult to find another value that hits exactly that target[3].

2.2 Data structures in blockchain

For storing information about transactions or accounts in a blockchain network, data structures such as *Merkle trees* (subsection 2.2.1) or *Bloom filters* (subsection 2.2.2) for finding logs are used.

2.2.1 Merkle trees

In cryptography and computer science, a *Merkle tree* (or a hash tree) is a tree in which every leaf node is labelled with the hash of a data block, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes.

Hash trees allow efficient and secure verification of the contents of large data structures[4].

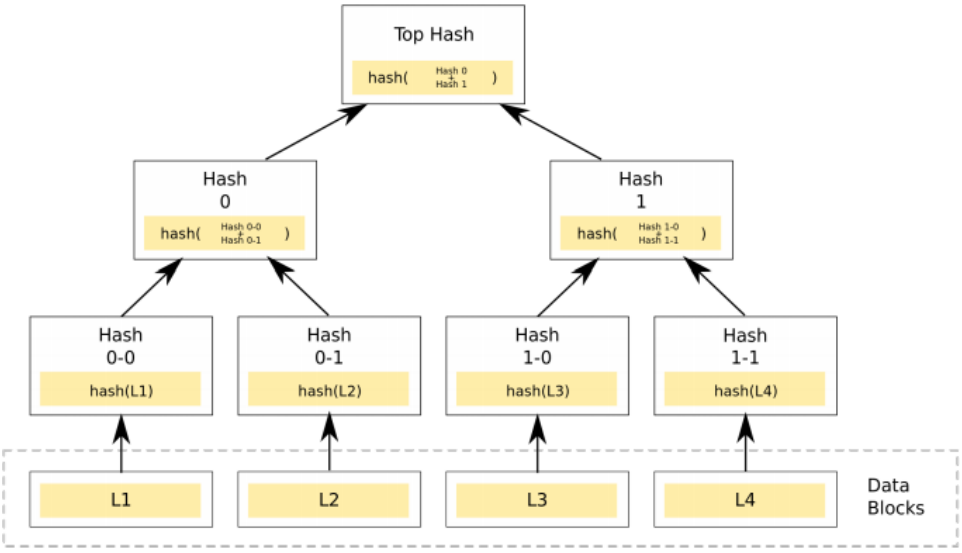

Figure 2.2: A binary Merkle (hash) tree

An example (Figure 2.2) of a binary hash tree. Hashes 0-0 and 0-1 are the hash values of data blocks L1 and L2, respectively, and hash 0 is the hash of the concatenation of hashes 0-0 and 0-1[5].

2.2.2 Bloom filter

A **Bloom filter** is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. *False positive* matches are possible, but false *negatives are not* – in other words, a query returns either "*possibly in set*" or "*definitely not in set*". Elements can be added to the set, but not removed; the more elements that are added to the set, the larger the probability of false positives[6].

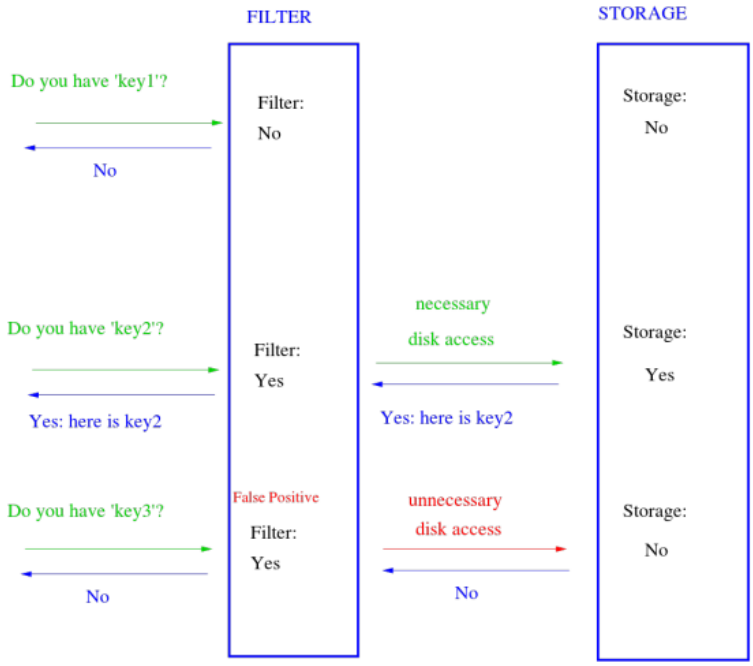

Figure 2.3: Example of a Bloom filter

While risking false positives, Bloom filters have a strong space advantage over other data structures for representing sets, such as self-balancing binary search trees, tries, hash tables, or simple arrays or linked lists of the entries.

Most of these require storing at least the data items themselves, however, Bloom filters do not store the data items at all, and a separate solution must be provided for the actual storage[7].

Ethereum uses *Bloom filters* for finding logs on the blockchain.

2.3 Three pillars of blockchain

The idea of blockchains is to provide a *decentralized*, *transparent*, and *immutable* technology that can record transactions between two parties efficiently and in a verifiable and permanent way[8].

2.3.1 Decentralization

The decentralized nature of the blockchain technology means that it does not rely on a central point of control which makes the system fairer and more secure.

Consensus protocols ensure valid transactions and record data in a manner that is incorruptible.

Consensus Protocol

A *consensus protocol* describes how the communication of data between nodes works. Consensus is achieved when enough devices are in agreement about what is true and should be recorded onto a blockchain[9].

2.3.2 Transparency

The transparency of a blockchain stems from the fact that the holdings and transactions of each public address are open to viewing.

Knowing a user's *public address*, it is possible to view their holdings and the transactions that they have carried out. However *linking public* addresses to individual users is particularly difficult to achieve[10].

2.3.3 Immutability

Immutability means that once something has been entered into the blockchain, it cannot be tampered with. Figure 2.4 showcases that even the slightest change in the input has enormous consequences on the output hash.

Tampering with any of the already linked block in the blockchain would break the *hash pointer* to the previous block[11].

INPUT	HASH
This is a test	C7BE1ED902FB8DD4D48997C6452F5D7E509FBCDBE2808B16BCF4EDCE4C07D14E
this is a test	2E99758548972A8E8822AD47FA1017FF72F06F3FF6A016851F45C398732BC50C

Figure 2.4: Cryptographic hashes of strings using SHA-256

2.4 Blockchain use cases

Cryptocurrency

Problem: International transfers using existing payment methods can be expensive and time consuming due to fees taken by middlemen.

Cryptocurrency is a digital currency that functions as a medium of exchange in buying and selling goods and services. Blockchain can facilitate peer-to-peer transactions conducting rapid and inexpensive cross-border payments where middlemen need no longer be involved.

Supply Chain Management

Problem: There is a lack of transparency as a product moves along its supply chain, which means that any problems that occur cannot adequately be isolated and investigated.

A supply chain is a network that is established between a business and its suppliers. Blockchain's ability to enable the digitization of assets would tag and assign unique identities to products that are then transplanted onto a transparent and immutable blockchain.

Digital Identity

Problem: Ownership of data is largely in the hands of the applications and services that are susceptible to data breaches and identity theft.

Blockchain could solve problems with digital identity, by giving control back to the user, so called *self-sovereign identity*. Individuals could possess an *encrypted digital hub* in which their digital identity data could be stored allowing individuals to control who had access.

Voting

Problem: Voter fraud and physical presence can contribute to an overall lowering of voter turnout.

The characteristics of blockchain make it a suitable candidate in addressing concerns of security and fraud. Blockchain can eliminate concerns of voter fraud by providing a clear record of votes that are cast. Hacking a blockchain enabled voting system would also be a difficult task, due to its tamper-proof characteristics.

Healthcare

Problems: Practitioners often do not have full access to a patient's medical history, which can make providing healthcare difficult.

Blockchain could serve as a secure and tamperproof database on which patient medical records can be stored. This would make it significantly easier for medical practitioners to gain a better understanding of a patient's medical history, by being able to access information such as medical drugs that the patient has taken in the past.[12].

Distributed Cloud Storage

Improvement in data security stemming from a shift from a centralized means of storage to a decentralized one.

Automobiles

Blockchain enabled supply chain management could prove useful in detailing car history.

Internet of Things

IoT devices, and all corresponding data can be stored on a decentralized blockchain, instead of being controlled by a centralized entity.

Insurance

Blockchain based smart contracts can produce a more transparent and secure arrangement between insurers and customers.

2.5 Blockchain working fundamentals

Inserting a new record to the public ledger requires validation called *mining* (subsection 2.5.1) utilizing *smart contracts* (subsection 2.5.2).

2.5.1 Mining

Mining in regards of cryptocurrencies is a *process of validation* of transactions and creating a new valid block. **Miner** is a node which is mining. By mining, new **coins** could be produced. These mined coins are a reward for the miner which mined a new block. It is a compensation and motivation for a computationally expensive process.

Miners can also earn coins from transaction fees. These are provided from transactions which are validated. The goal is to completely cover mining costs with transition fees. Providing *proof* is part of mining as well.

Specialized machines such as FPGAs[1] and ASICs[2] running complex hashing algorithms (e.g. SHA-256) have been created specifically for mining purposes[13].

2.5.2 Smart contracts

A *smart contract* is a computerized transaction protocol that executes the terms of a *contract* without the need for a trusted third party[14]. A blockchain-based smart contract is visible to all users of a blockchain.

Smart contracts can[15]:

- Function as "multi-signature" accounts, so that funds are spent only when a required percentage of people agree.
- Manage agreements between users, say, if one buys insurance from the other.
- Provide utility to other contracts (similar to how a software library works).
- Store information about an application, such as domain registration information or membership records.

1. Field-programmable gate array
2. Application-specific integrated circuit

2.6 Proof systems

Byzantine Fault Tolerance

Byzantine Fault Tolerance (BFT) is the characteristic which defines a system that tolerates the class of failures that belong to the *Byzantine Generals Problem*. Byzantine Failure is the most difficult class of failure modes as it implies no restrictions, and makes no assumptions about the kind of behavior a node can have (e.g. a node can generate any kind of arbitrary data while posing as an honest actor).

Byzantine Generals Problem

A commanding general must send an order to his *n - 1* lieutenant generals such that:

IC1. All loyal lieutenants obey the same order.

IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Conditions IC1 and IC2 are called the interactive consistency conditions. Note that if the commander is loyal, then IC1 follows from IC2. However, the commander need not be loyal.

Adding to IC2, if the commander is a traitor, consensus must still be achieved. As a result, all lieutenants take the majority vote. The algorithm to reach consensus in this case is based on the value of majority of the decisions a lieutenant observes.

Theorem 1. *For any m, Algorithm OM(m) reaches consensus if there are more than 3m generals and at most m traitors[16].*

This implies that the algorithm can reach consensus as long as $\frac{2}{3}$ of the actors are honest. If the traitors are more than $\frac{1}{3}$, consensus is not reached.

Blockchains are decentralized ledgers which, by definition, are not controlled by a central authority. Due to the value stored in these ledgers, bad actors have huge economic incentives to try and cause faults. In the absence of BFT, a peer is able to transmit and post false transactions effectively nullifying the blockchain's reliability.

At the time of invention of *Bitcoin*, the use of **Proof of Work** as a probabilistic solution to the *Byzantine Generals Problem*.

2.6.1 Proof of Work

The purpose of the *Proof of Work* is to prove that a particular amount of computation has been expended to generate some output (i.e. *nonce*). This is because there is no better way to find the *nonce* that is below the required threshold other than to enumerate all the possibilities.

The outputs of repeatedly applying the hash function have a uniform distribution, and so, on average, the time needed to find such a *nonce* depends on the *difficulty*[3]. The higher the difficulty, the longer it takes to solve for the *nonce*. The *PoW* algorithm gives meaning to the concept of difficulty, which is used to enforce blockchain security.

PoW ensures that a particular blockchain remains canonical, making it computationally difficult for an attacker to create new blocks that overwrite a part of history (e.g. by erasing transactions) or maintaining a fork. To have their block validated first, an attacker would need to consistently solve for the *nonce* faster than anyone else in the network. This would be impossible unless the attacker had more than half of the network's *mining power*, known as the **Majority attack**[17].

Proof of Work example

Let us assume the base string to work on is "*Hello, world!*." The target is to find a variation of it that SHA-256 hashes to a value smaller than 2^{240}. Varying the string by adding an integer value to the end called a *nonce* and incrementing it each time, then interpreting the hash result as a long integer and checking whether it is smaller than the target 2^{240}. Finding a match for "Hello, world!" takes 4251 tries and showcases that finding the right solution takes a significant amount of computation compared checking the 'correctness' of the final hash[18].

```
"Hello, world!0" => 1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64 = 2^252.253458683
"Hello, world!1" => e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8 = 2^255.868431117
"Hello, world!2" => ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4266b7 = 2^255.444730341
...
"Hello, world!4248" => 6e110d98b388e77e9c6f042ac6b497cec46660deef75a55ebc7cfdf65cc0b965 = 2^254.782233115
"Hello, world!4249" => c004190b822f1669cac8dc37e761cb73652e7832fb814565702245cf26ebb9e6 = 2^255.585082774
"Hello, world!4250" => 0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9 = 2^239.61238653
```

Figure 2.5: "Hello World!" Proof of Work example

3. Difficulty is a measure of how difficult it is to find a hash below a given target.

2.6.2 Other proof mechanisms

Proof of Stake

In *Proof of work*, if *Bob* has more computational power and energy than *Alice* and thus can output more work he is more likely to win (mine the next block).

Proof of Stake (PoS) takes away the energy and computational power requirement of PoW and replaces it with **stake**. *Stake* is referred to as an amount of currency that an actor is willing to lock up for a certain amount of time. In return, they get a chance proportional to their stake to be the next leader and select the next block.

The main issue with PoS is the so-called *nothing-at-stake* problem. Essentially, in the case of a fork, stakers are not disincentivized from staking in both chains, and the danger of *double spending* problems increase[19].

Proof of Space

Proof of Space, also called *Proof of Capacity (PoC)*, is a means of showing that one has a legitimate interest in a service by allocating a non-trivial amount of memory or disk space to solve a challenge presented by the service provider[20].

Proof of space was researched in the context of cryptocurrencies as an alternative to Bitcoin's Proof of Work system which consumes much more electricity compared to Proof of Space. Therefore, Proof of Space is recognized as more environmentally friendly[21].

Proof of Authority

Proof of Authority (PoA) is an algorithm used with blockchains that delivers comparatively fast transactions through a consensus mechanism based on identity as a stake. In this system, only predefined (authorized) nodes can create a new block. It's an evolution of Proof of Stake where the stake was replaced with identity.

The best use of Proof of Authority mechanism is in the private blockchains. It is the best option as transaction could be added almost instantaneously and without any high demand on storage, CPU which makes it a very efficient solution[22].

2.7 Common attacks on blockchains

As every technology, blockchains are susceptible to attack too. The most known attacks are the *majority attack* (subsection 2.7.1) and the *Sybil attack* (subsection 2.7.2).

2.7.1 Majority attack

The **majority**, or *51% attack* refers to an attack on a blockchain – by a group of miners controlling more than 50% of the network's mining **hashrate**, or *computing power*. The attackers would be able to prevent new transactions from gaining confirmations, allowing them to halt payments between some or all users[23]. They would also be able to reverse transactions that were completed while they were in control of the network, meaning they could double-spend coins[24].

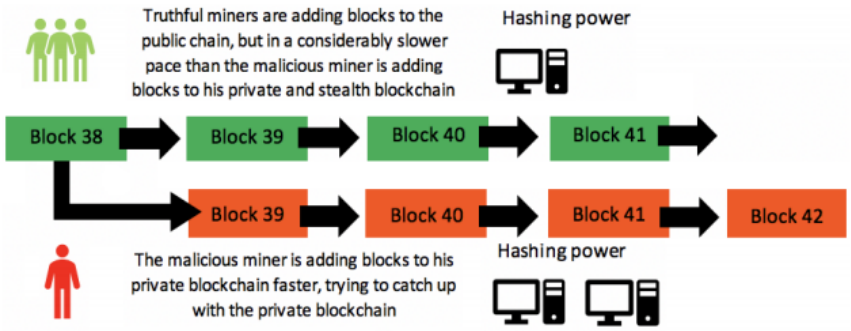

Figure 2.6: Visualization of the 51% attack

2.7.2 Sybil attack

Sybil attack is a type of attack seen in peer-to-peer networks in which a node in the network operates multiple identities actively at the same time and undermines the authority/power in reputation systems.

The main aim of this attack is to gain the majority of influence in the network to carry out illegal actions in the system. A single entity has the capability to create and operate multiple identities (user accounts, IP address based accounts). To outside observers, these multiple fake identities appear to be real unique identities[25].

2.8 Blockchain types

Currently, there are four types of blockchain networks — *public*, *private*, *consortium*, and *hybrid* blockchains.

2.8.1 Public blockchain

Public blockchain protocols are based on the *Proof of Work* consensus algorithms and are available as open source. Anyone can: participate in these without permission, download the code, run any of the public nodes on their local device to validate transactions in the network.

Anyone in any part of the world can send their transactions through this network, can expect to see them included in the blockchain if they are valid, read the transaction on a public block explorer. All the transactions are transparent but anonymous.

2.8.2 Consortium blockchain

Consortium blockchains always operate under the control of a *group*. In comparison to public blockchains, they do not allow just anyone with Internet access to participate in the transaction verification.

They are faster, more scalable and even provide more transaction privacy. The consensus process is in the control of a pre-selected set of nodes. The right to read the complete blockchain may be public, or may be restricted only to the pre-selected participants.

2.8.3 Private blockchain

Private blockchains contain different groups and participants who can easily verify various transactions internally. Here, write permissions are kept centralised within just one organisation whereas read permissions might be public or restricted to a certain extent[26].

2.8.4 Hybrid blockchain

A hybrid blockchain has a combination of centralized and decentralized features. The exact workings of the chain can vary based on which portions of centralization and decentralization are used[27].

2.9 Blockchain qualitative analysis

In the context of a qualitative analysis of open-source blockchain technologies, the most common criteria are as follows[28]:

- Activity - How active is the blockchain platform in terms of blockchain development projects?
- Type of network - Whether a blockchain platform supports public, permissioned or private network.
- Supported Languages - What are the programming languages supported by the blockchain platform's SDK?
- Popularity - How popular is the platform based on its contribution to GitHub and ratings?
- Costing - If the platform is available at free of cost or if it is paid, then what is its price per year.
- Technical Support - If the platform is backed a team of technical support specialists or not.

The most popular open-source blockchain technologies, in no particular order, are[29][30]:

1. Ethereum
2. Hyperledger
3. Multichain
4. Corda
5. Open-chain
6. MultiChain
7. BigchainDB
8. HydraChain
9. EOS
10. Stellar

2.10 Blockchain testing

Once a block is added to the blockchain, it cannot be changed, and if any data on it is modified, the following blocks become invalid. This makes it crucial to ensure that blocks are being added in the right way. Testing blockchains is both critical and complex.

Challenges

Validating and verifying the adoption of blockchain comes with a number of inherent challenges because there is a significant change in the technology itself. In addition to standard testing and validation such as functional testing, non-functional testing, performance testing, security testing and integration testing, testing teams also need specialized testing capabilities, such as *Smart Contract Testing* and *Peer/Node Testing*[31]. Common challenges include:

- Lack of best practices - Learning skills or understanding the concepts blockchain applications is expensive.
- Lack of standardization in blockchain testing - Developers without a strong ability to conceptualize, standardize, and abstract complex concepts in blockchain lead to a number of issues.
- Transactions are irreversible - Once a transaction occurred, it cannot be modified or deleted.
- Security - The applications are vulnerable to attacks at network level, user level, and mining level.
- Lack of testing tools - Testing blockchain applications requires a mixed toolset.
- Block and chain size - Without proper validation of block size and chain size leads to failure of blockchain applications.
- Sub-optimal testing strategy - Testing is given less significance over programming, resulting in developing a blockchain application development environment with fewer or no dedicated testers to explore and evaluate the final product.

2.10.1 Types of blockchain testing

- Functional testing - This tests that the system's components are working as required.
- Integration testing - The interfaces between the components, the integrations, and the different parts of the system need to function properly together. This is essential to ensure consistent performance.
- Security testing - This needs to check if the application is vulnerable to attacks, if authorization systems are robust, whether adequate data protection is in place, and there is no vulnerability to malicious attacks.
- Performance testing - The performance of an application and the latency vary based on the network and transaction size. Performance testing identifies performance bottlenecks, and finds ways to overcome them[32].

3 Ethereum

The Ethereum blockchain is a **transaction-based state machine**. It begins with a *genesis state* which is analogous to a blank slate, before any transactions have happened on the network. When transactions are executed, this genesis state transitions into some final state. At any point in time, this final state represents the current state of Ethereum.

To cause a transition from one state to the next, a transaction must be valid. For a transaction to be considered valid, it must go through a validation process known as **mining**. Mining is when a group of nodes (i.e. computers) expend their compute resources to create a block of valid transactions.

Any node on the network that declares itself as a *miner* can attempt to create and validate a block. Each miner provides a *mathematical proof* when submitting a block to the blockchain, and this proof acts as a guarantee: if the proof exists, the block must be valid.

For a block to be added to the main blockchain, the miner must prove it faster than any other competitor miner. The process of validating each block by having a miner provide a **Proof of Work**.

A miner who validates a new block is rewarded with a certain amount of value - an intrinsic digital token called **Ether** (ETH). Every time a miner proves a block, new Ether tokens are generated and awarded[33].

The Ethereum system is comprised of:

- Accounts and States
- Gas and fees
- Transactions and messages
- Blocks
- Transaction execution
- Mining
- Proof of Work

3.1 Accounts and States

Ethereum is comprised of *accounts* that are able to interact with one another and have a **state** (subsection 3.1.1) associated with them.

There are two types of accounts:

- Externally owned accounts, which are controlled by *private keys* and have no code associated with them.
- Contract accounts, which are controlled by their *contract code* and have code associated with them.

An externally owned account can send messages by creating and signing a transaction using its *private key*. A message between two externally owned accounts is a *value transfer*. A message from an externally owned account to a contract account activates the contract account's code, allowing it to perform various actions (e.g. transfer tokens, perform some calculations, etc.).

Contract accounts cannot initiate new transactions, only trigger transactions in response to other transactions they have received. Therefore, any action that occurs on the Ethereum blockchain is always initiated by transactions from externally controlled accounts.

3.1.1 Account state

The account state consists of four components, which are present regardless of the account type:

- **nonce** - Either the number of transactions sent from an externally owned account's address or the number of contracts created by a contract account.
- **balance** - The account balance in *Wei* (1 ETH = $1E^{18}$ Wei).
- **storageRoot** - The hash of the root node (subsection 2.2.1).
- **codeHash** - The hash of the EVM code (subsection 3.5.3).

3.1.2 World state

The global state consists of a key/value mapping between the *account addresses* and the *account states*[34].

3.2 Gas and fees

Every computation that occurs as a result of a transaction on the Ethereum network incurs a **fee** which is paid in a denomination called **gas**. Gas is the unit used to measure the fees required for a particular computation.

Gas price is the amount of Ether the sender is willing to spend on every unit of gas. With every transaction, a sender sets a *gas limit* and *gas price* (Figure 3.1) that represent the maximum amount that the sender is willing to pay for executing a transaction.

All the resources spent on gas by the sender is sent to the *beneficiary* address, which is typically the miner's address. Since miners are expending the effort to run computations and validate transactions, miners receive the gas fee as a reward[35].

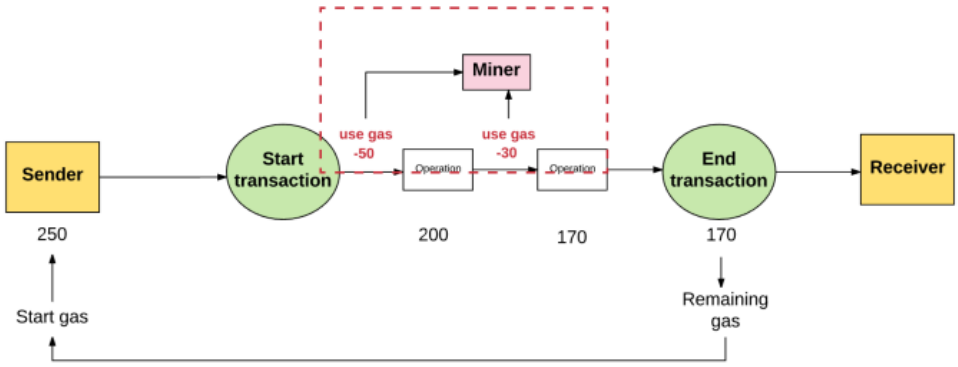

Figure 3.1: Gas spent on operations during a transaction

Ethereum is a *Turing complete* language. This allows for loops and makes Ethereum susceptible to the *halting problem*. If there were no fees, a malicious actor could easily try to disrupt the network by executing an infinite loop within a transaction, without any repercussions. Thus, fees protect the network from deliberate attacks.

3.3 Transactions and messages

In the most basic sense, a transaction is a *cryptographically signed* piece of instruction that is generated by an externally owned account, *serialized*, and then *submitted to the blockchain*.

There are two types of transactions[36]:

- **Messages** - A message can be a transaction signed by an externally owned account or from a contract account to another contract account.

- **Contract creations** - Contracts are virtual objects that, unlike transactions, are not serialized and only exist in the Ethereum execution environment. When one contract sends an internal transaction to another contract, the associated code that exists on the recipient contract account is executed.

3.4 Blocks

All transactions are grouped together into *blocks*. A blockchain contains a series of such blocks that are chained together.

In Ethereum, a block consists of:

- The block header.

- Information about the set of transactions included in that block.

- A set of other block headers for the current block's *ommers*.

An *ommer* is a block whose parent is equal to the current block's parent's parent. In Ethereum, block times are much lower than those of other blockchains. This enables faster transaction processing, however, the downside of shorter block times is that more competing block solutions are found by miners.

These competing blocks are also referred to as *orphaned blocks* (i.e. mined blocks do not make it into the main chain) for which miners receive a smaller reward than a full block[37].

3.4.1 Block header

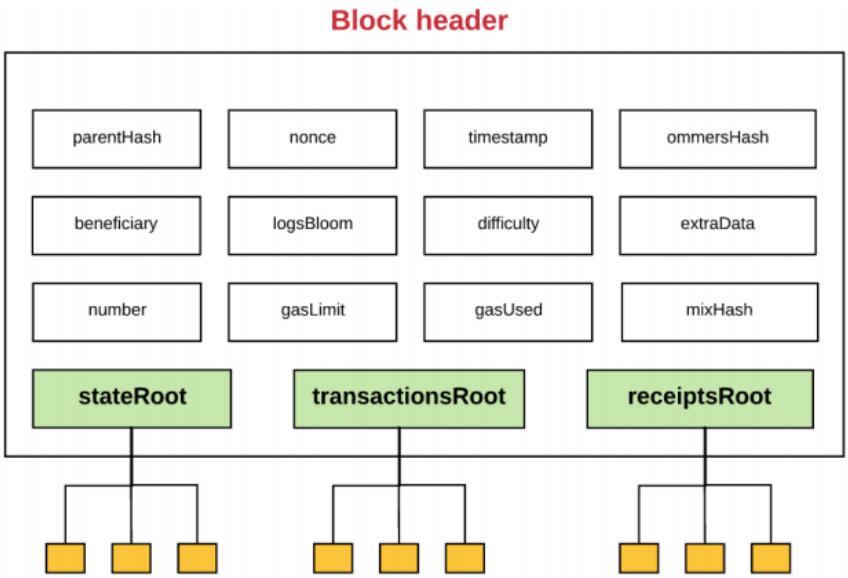

Figure 3.2: Block header contents

A block header is a part of the block that, among other fields (see Figure 3.2), consists of[38]:

- **logsBloom** - A log entry contains:
  - The logger's account address.
  - A series of topics that represent various events carried out by this transaction.
  - Any data associated with these events.
- **difficulty** - The *difficulty* of a block is used to enforce consistency in the time it takes to validate blocks. If a certain block is validated more quickly than the previous block, the Ethereum protocol increases that block's difficulty.

  The difficulty of the block affects the nonce, which is a hash that must be calculated when mining a block, using *Proof of Work*.

3.5 Transaction Execution

Executing a transaction in Ethereum requires a series of steps that are described in details in subsection 3.5.1 to subsection 3.5.4.

3.5.1 Transaction requirements

First, all transactions must meet an initial set of requirements in order to be executed. These include:

- The transaction must be a properly formatted **RLP** (Recursive Length Prefix) which is a data format used to encode nested arrays of binary data. RLP is the format Ethereum uses to serialize objects.
- Valid transaction signature.
- Valid transaction *nonce*. Recall that the nonce of an account is the count of transactions sent from that account. To be valid, a transaction nonce must be equal to the sender account's nonce.
- The transaction's gas limit must be equal to or greater than the intrinsic gas used by the transaction. The intrinsic gas includes:
  1. A predefined cost of 21 000 gas for executing a transaction.
  2. A gas fee for data sent with the transaction:
     - 4 gas for every byte of data or code that equals zero.
     - 68 gas for every non-zero byte of data or code.
  3. If the transaction is a contract-creating transaction, an additional 32 000 gas.

The sender's account balance must have enough Ether to cover the *upfront gas costs* that the sender must pay.

The calculation for the upfront gas cost is the following:

1. The transaction's gas limit is multiplied by the transaction's gas price to determine the maximum gas cost.
2. This maximum cost is added to the total value being transferred from the sender to the recipient.

3.5.2 Execution process

First, the upfront cost of execution from the sender's balance is deducted, and the *nonce* of the sender's account is increased by 1 to account for the current transaction. At this point, the the remaining gas is the *total gas limit* for the transaction **minus** the *intrinsic gas*.

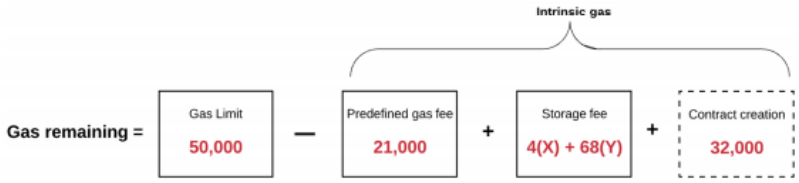

Figure 3.3: Remaining gas for a transaction

Next, the transaction starts executing. Throughout the execution of a transaction, Ethereum keeps track of the **substate** which is a way to record information accrued during the transaction. It contains:

- **Self-destruct set** - A set of accounts (if any) that will be discarded after the transaction completes.
- **Log series**: Archived and indexable checkpoints of the virtual machine's code execution.
- **Refund balance** - The amount to be refunded to the sender account after the transaction.

Next, the computations required by the transaction are processed and the state is finalized by determining the amount of unused gas to be refunded to the sender. Once the sender is refunded:

- The Ether for the gas is given to the miner.
- The gas used by the transaction is added to the *gas counter* (which keeps track of the total gas used by all transactions in the block).
- All accounts in the *self-destruct set* (if any) are deleted.

Finally, the global state is left with the new state and a set of the logs created by the transaction[39].

3.5.3 Ethereum Virtual Machine

The part of the protocol that handles the processing of the transactions is the **Ethereum Virtual Machine** (EVM) which is *Turing complete* with a limitation being that the EVM is intrinsically bound by gas. Thus, the total amount of computation that can be done is limited by the amount of gas provided.

Moreover, the EVM has a **stack-based architecture** that uses a last-in, first-out *stack* to hold temporary values in a *volatile* memory.

The EVM also has a *non-volatile* storage that is maintained as part of the system state. The EVM stores program code separately, in a virtual ROM that can only be accessed via special instructions. In this way, the EVM differs from the typical von Neumann architecture, in which program code is stored in memory or storage[40].

3.5.4 Block finalization

Block finalization can mean two different things, depending on whether the block is new or existing.

1. If it is a new block, it refers to the process required for mining.
2. If it is an existing block, it refers to the process of validation.

In either case, there are four requirements for a block to be *finalized*:

1. **Validate (or, if mining, determine) ommers** - Every *ommer* block within the block header must be a valid header.
2. **Validate (or, if mining, determine) transactions** - The *gasUsed* number on the block must be equal to the cumulative gas used by the transactions listed in the block.
3. **Apply rewards (only if mining)** - The beneficiary address is awarded 3 Ether for mining the block and an additional $\frac{1}{32}$ of the current block reward for each *ommer*.Lastly, the beneficiary of the *ommer block(s)* also gets awarded a certain amount.
4. **Verify (or, if mining, compute a valid) state and nonce** - Ensure that all transactions and resultant state changes are applied. Verification occurs by checking this final state against the state tree stored in the header.

3.6 Mining Proof of Work

Ethereum's Proof of Work algorithm is called **Ethash**[1].

A **seed** is calculated for each block. This seed is different for every **epoch** where each epoch is 30 000 blocks long. For the first epoch, the seed is the hash of a series of 32 bytes of zeros. For every subsequent epoch, it is the hash of the previous seed hash. Using this seed, a node can calculate a pseudo-random **cache**.

This cache enables the concept of **light nodes**, the purpose of which is to afford certain nodes the ability to efficiently verify a transaction without the need to store the entire blockchain dataset. A light node can verify the validity of a transaction based solely on this cache, because the cache can regenerate the specific block it needs to verify.

Using the cache, a node can generate the *DAG dataset*, where each item in the dataset depends on a small number of pseudo-randomly selected items from the cache. In order to be a miner, you must generate this full dataset; all full clients and miners store this dataset, and the dataset grows linearly with time.

Miners can then take random slices of the dataset and put them through a mathematical function to hash them together into a *mixHash*. A miner will repeatedly generate a *mixHash* until the output is below the desired target *nonce*. When the output meets this requirement, this nonce is considered valid and the block can be added to the chain[41].

1. previously known as Dagger-Hashimoto

4 Fabric

Hyperledger Fabric is an **enterprise-grade permissioned distributed ledger framework**[42] and it is one of the blockchain projects within Hyperledger. Like other blockchain technologies, it has a *ledger*, uses *smart contracts*, and is a system by which participants manage their transactions.

Where Hyperledger Fabric differs from some other blockchain systems is that it is **private** and **permissioned** where members of a network enroll through a trusted MSP (Membership Service Provider).

Ledger data can be stored in multiple formats, consensus mechanisms can be swapped in and out, and different MSPs are supported.

4.1 Shared Ledger

Hyperledger Fabric has a ledger subsystem comprising two components: **the world state** and **the transaction log**.

- The *world state* component describes the state of the ledger at a given point in time. It is the database of the ledger.
- The *transaction log* component records all transactions which have resulted in the current value of the world state; it is the update history for the world state.

The ledger is a combination of the world state database and the transaction log history and each participant has a copy of the ledger to every network they belong to.

4.2 Privacy

Depending on the needs of a network, Hyperledger Fabric supports networks where privacy using **channels** is a key operational requirement as well as networks that are comparatively open.

4.2.1 Channels

Channels can be defined as a logical entity which represents a grouping of two or more network members or participants for the purpose of conducting private and confidential transactions between themselves[43].

4.3 Chaincode

Hyperledger Fabric smart contracts are written in **chaincode** and are invoked by an application external to the blockchain when that application needs to interact with the ledger.

In most cases, chaincode interacts only with the database of the ledger and not the transaction log.

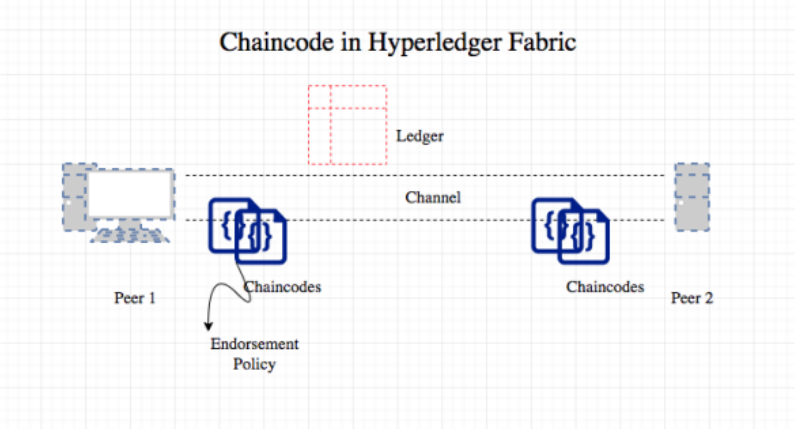

Figure 4.1: Chaincode visualization

4.3.1 Endorsement policy

Every chaincode has an *endorsement policy*, which specifies the set of peers on a channel that must execute chaincode and endorse the execution results in order for the transaction to be considered valid. These endorsement policies define the organisation's peers who must endorse, i.e., approve of the execution of a proposal[44].

4.4 Peers

So not all peer nodes are the same. There are different types of peer nodes with different roles in the network[45]:

- Endorser peer
- Anchor peer
- Orderer peer

4.4.1 Endorser peer

Peers can be marked as *Endorser peer*. Upon receiving the **transaction invocation request** from the client application the endorser peer:

1. Validates the transaction (i.e., check certificate details and roles of the requester).
2. Executes the *chaincode* and simulates the outcome of the transaction but it does not update the ledger.

As **only** the *endorser* node executes the *chaincode* (smart contract), there is no necessity to install chaincode in each and every node of the network which increases the scalability of the network.

4.4.2 Anchor peer

Anchor peer or cluster of anchor peers is configured at the time of the *channel* configuration.

Anchor peers receive updates and broadcast the updates to the other peers in the organization. Anchor peers are discoverable. So any peer marked as *anchor peer* can be discovered by the *orderer peer* or any other peer.

4.4.3 Orderer peer

Orderer peer is considered as the central communication channel for the Hyperledger Fabric network. Orderer peer/node is responsible for consistent ledger state across the network. *Orderer peer* creates the block and delivers that to all the peers.

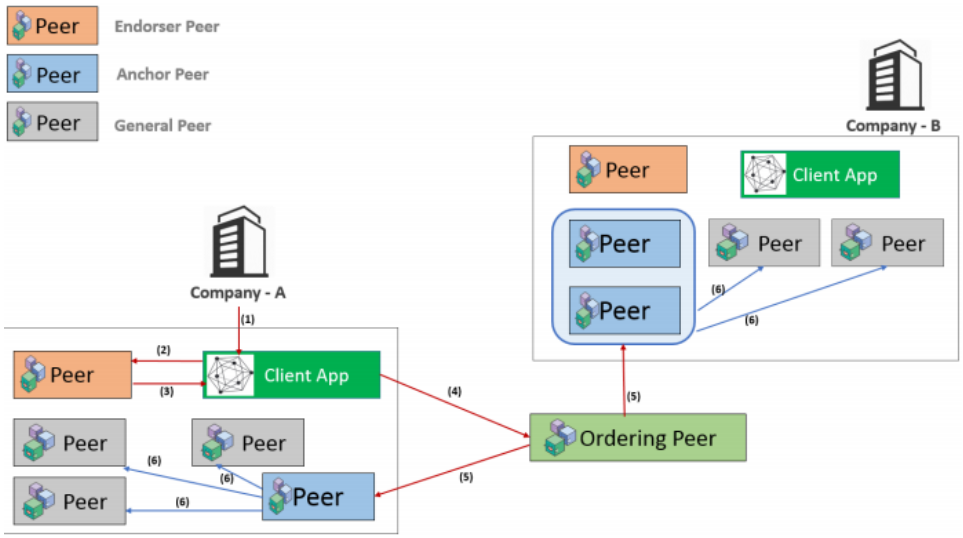

Figure 4.2: Hyperledger Fabric workflow

4.5 The Ordering Service

Open, unpermissioned blockchains where any node can participate in the consensus process rely on probabilistic consensus algorithms. These algorithms guarantee ledger consistency to a high degree of probability but are still vulnerable to divergent ledgers - *forks*.

Hyperledger Fabric features a kind of a node called an **orderer** (also known as an *ordering node*) that does transaction ordering, which along with other nodes forms an **ordering service**.

Because Fabric's design relies on deterministic consensus algorithms, any block a peer validates as generated by the *ordering service* is guaranteed to be **final** and **correct**. Ledgers cannot fork the way they do in open distributed blockchains.

While the *ordering service* is **central** to the Fabric network, it can be made of several nodes that work together to perform the ordering function. Fabric is thus designed so that the ordering service can be decentralized and can provide for a Fabric network to be fully decentralized as expected from a blockchain framework[46].

4.6 Orderers and the transaction flow

Applications that want to update the ledger are involved in a process with *three* phases that ensures all of the peers in a blockchain network keep their ledgers consistent with each other.

4.6.1 Proposal

In the first phase, a client application sends a *transaction proposal* to a subset of peers that will invoke a smart contract to produce a proposed ledger update and then endorse the results.

The endorsing peers do not apply the proposed update to their copy of the ledger at this time. Instead, the endorsing peers return a proposal response to the client application.

4.6.2 Ordering and packaging transactions into blocks

After a client application has received an endorsed transaction proposal response from a set of peers, the client submits the transactions containing endorsed transaction proposal responses to an ordering service node.

Ordering service nodes receive transactions from many different application clients concurrently. These ordering service nodes work together to collectively form the **ordering service**. Its job is to arrange batches of submitted transactions into a well-defined sequence and package them into blocks. The blocks are then saved to the orderer's ledger and distributed to all peers in channel.

4.6.3 Validation and commit

Lastly, each peer will validate distributed blocks independently to ensure it has been endorsed by the required organization's peers, that its endorsements match, and that it has not become invalidated by other recently committed transactions which may have been in-flight when the transaction was originally endorsed.

Invalidated transactions are still retained in the immutable block created by the orderer, but they are marked as invalid by the peer and do not update the ledger's state[47].

4.7 Ordering service implementations

The ordering service can have multiple implementations. *Solo* (subsection 4.7.1) is used in development environments and *Kafka* (subsection 4.7.3) is used in production environments.

4.7.1 Solo

The Solo implementation of the ordering service features only a **single** ordering node. As a result, it is not fault tolerant. *Solo* is designed for testing applications and smart contracts.

4.7.2 Raft

Raft is a CFT[1] ordering service. It follows a *leader and follower* model, where a **leader** node is elected (per channel) and its decisions are replicated by the **followers**. Raft's design allows different organizations to contribute nodes to a distributed ordering service.

4.7.3 Kafka

Kafka is a **distributed, horizontally-scalable, fault-tolerant, commit log** - a data structure that only appends. No modification or deletion is possible, and the worst case complexity is $\mathcal{O}(1)$.

There can be multiple Kafka nodes in the blockchain network, with their corresponding **Zookeeper Service** ensemble[48].

4.8 Crash tolerance

The *crash-tolerance* mechanism replicates the partitions among the multiple Kafka brokers. Thus if one broker dies, due to a software or a hardware fault, data is preserved.

What follows is a **leader-follower system**, wherein the leader owns a partition, and the follower has a replication of the same partition. When the leader dies, the follower becomes the new leader. If a consumer is subscribing to a topic, the **Zookeeper service** determines from which leader to read the subscribed messages.

1. Crash fault tolerant

4.9 The Zookeeper Service

Zookeeper is a *distributed key-value store*, most commonly used to store metadata and handle the mechanics of clustering. It allows clients of the service (the Kafka brokers) to subscribe and have changes sent to them once they happen. This is how brokers know when to switch partition leaders.

4.10 Kafka in Hyperledger Fabric

Each channel maps to a separate single-partition topic in Kafka. When an OSN[2] receives transactions, it checks to make sure that the broadcasting client has permissions to write on the channel, then relays (i.e. produces) those transactions to the appropriate partition in Kafka.

This partition is also consumed by the OSN which groups the received transactions into blocks locally, persists them in its local ledger, and serves them to receiving clients[49].

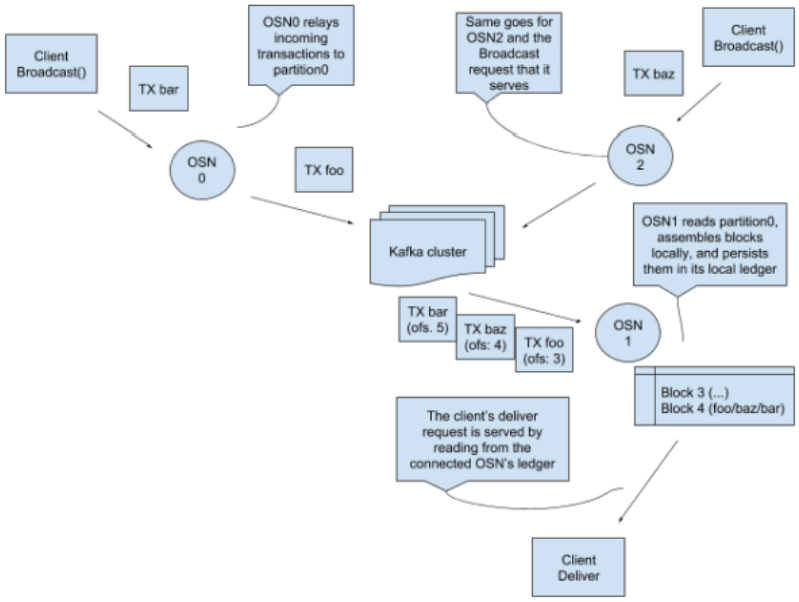

Figure 4.3: Kafka working example

2. Ordering Service Node

5 Besu

Hyperledger Besu is an open-source Ethereum client which can be run both on the Ethereum public network or on private permissioned networks and test networks such as *Rinkeby*, *Gorli*, and *Ropsten*.

Besu helps develop enterprise applications that require secure, high-transaction processing in a private network by implementing permissioning and privacy[50].

The key three core components of the Besu are the following[51]:

- Storage
- Ethereum Core
- Networking

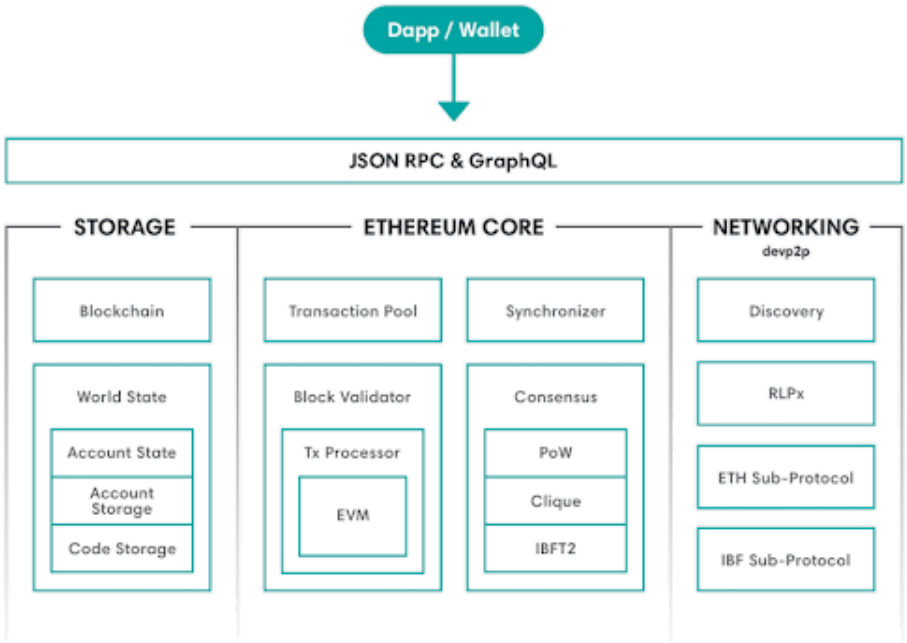

Figure 5.1: Besu architecture

5.1 Consensus protocols

Besu implements a number of consensus protocols[52]:

1. Proof of Work - Ethash
2. Proof of Authority
   (a) **Clique** - In Clique networks, transactions and blocks are validated by approved accounts, known as signers.
   (b) **IBFT 2.0** - In IBFT 2.0 networks, transactions and blocks are validated by approved accounts, known as validators.

5.1.1 Clique vs. IBFT 2.0

Properties to consider when comparing Clique and IBFT 2.0 are[53]:

Immediate finality

In IBFT 2.0, there's only 1 block proposed at a given chain height, there are no forks and all valid blocks are included in the main chain.

Clique does not have immediate finality. Implementations using Clique must be aware of forks and chain reorganizations occurring.

Minimum number of validators

IBFT 2.0 requires 4 validators to be Byzantine fault tolerant. Clique can operate with a single validator but operating with a single validator offers no redundancy if the validator fails[54].

Fault tolerance

Clique tolerates up to half to the signers failing. IBFT 2.0 networks require greater than or equal to $\frac{2}{3}$ of validators to be operating.

Speed

Reaching consensus and adding blocks is faster in Clique networks but the probability of a fork increases with the number of validators.

For IBFT 2.0, the time to add new blocks increases as the number of validators increases.

5.2 Privacy

Privacy in Hyperledger Besu refers to the ability to keep transactions private between the involved parties. Other parties cannot access the transaction content, sending party, or list of participating parties[55].

5.2.1 Private Transaction Manager

Besu uses a Private Transaction Manager to implement privacy, for example, *Orion*. Each Besu node that sends or receives private transactions requires an associated Orion node.

Besu and Orion nodes both have public/private key pairs identifying them. The private transaction submitted from the Besu node to the Orion node is signed with the Besu node private key.

5.2.2 Privacy groups

A privacy group is a group of nodes identified by a unique privacy group ID by Orion. Each private transaction is stored in Orion with the privacy group ID.

The Besu nodes maintain the public world state for the blockchain and a private state for each privacy group. The private states contain data that is not shared in the globally replicated world state.

5.3 Permissioning

Besu uses node permissioning to restrict access to known participants.

5.3.1 Local

Local permissioning is specified at the node level where each node in the network has a permissions configuration file which affects the node but not the rest of the network.

5.3.2 Onchain

Onchain permissioning is specified in a smart contract on the network. Specifying permissioning onchain enables all nodes to read and update permissioning configuration from one location.

5.4 Transactions

When it comes to transactions, Besu only implements *restricted* transactions only. This is done to ensure privacy and make all transactions private.

All nodes maintain a transaction pool where pending transactions are stored before they are processed.

5.4.1 Validating transactions

For a transaction to be considered valid, it has to go through a validation process by the *Tx Validation Pool*. The requirements are[56]:

1. Nonce is high enough
2. Permissions are correct
3. Sender is valid
4. Account balance is sufficient
5. Chain ID is correct
6. Gas limit is high enough

When the transaction is added to a block an additional validation is performed to check the transaction gas limit is less than the remaining block gas limit.

After creating a block, the node imports the block and the transaction pool validations are repeated.

6 Iroha

Hyperledger Iroha is a general purpose permissioned blockchain system that can be used to manage digital assets, identity, and serialized data. This can be useful for applications such as interbank settlement, central bank digital currencies, payment systems, national IDs, and logistics, among others[57].

6.1 Consensus mechanism

Hyperledger Iroha features a simple construction, emphasis on mobile application development and a new, chain-based Byzantine Fault Tolerant consensus algorithm, called **YAC** (Yet Another Consensus).

6.1.1 Yet another consensus

YAC is based on voting for block hash. Consensus involves taking blocks after they have been validated, collaborating with other blocks to decide on commit, and propagating commits between peers.

YAC algorithm achieves consensus through the following steps[58]:

1. The ordering service shares a **proposal** to all peers. A proposal is an unsigned block shared to peers in the network by the ordering service. It contains a batch of ordered transactions.

2. Peers calculate the hash of a verified proposal and sign it. The resulting `<Hash, Signature>` tuple is called a **vote**.

3. Based on the hashes created in the previous step, each peer computes an ordering list or order of peers. The first peer in the list is called the **leader** who is responsible for collecting votes from other peers and sending the commit message.

4. Each peer votes. The leader collects all the votes, determines the supermajority of votes for a certain hash and sends a **commit** message that contains the votes of the committing block.

5. After receiving the commit, the peers verify the commit and apply the block to the ledger and the consensus is complete.

6.2 Transactions

The transaction steps for Iroha are the following[59]:

1. A client sends a transaction to the *Torii* gate (an input and output interfaces for clients), which routes the transaction to a peer that is responsible for performing stateless validation.
2. After the peer performs *stateless validation*, the transaction is first sent to the ordering gate, which is responsible for choosing the right strategy of connection to the *ordering service*.
3. The ordering service puts transactions into order and forwards them to peers in the consensus network in form of proposals.
4. Each peer verifies the proposal's contents (stateful validation) in the *Simulator* (which generates a temporary snapshot of storage to validate transactions) and creates a block which consists only of verified transactions. This block is then sent to the consensus gate which performs YAC consensus logic.
5. An ordered list of peers is determined, and a leader is elected based on the YAC consensus logic. Each peer casts a vote by signing and sending their proposed block to the leader.
6. If the leader receives enough signed proposed blocks (i.e. more than two thirds of the peers), then it starts to send a commit message, indicating that this block should be applied to the chain of each peer participating in the consensus.

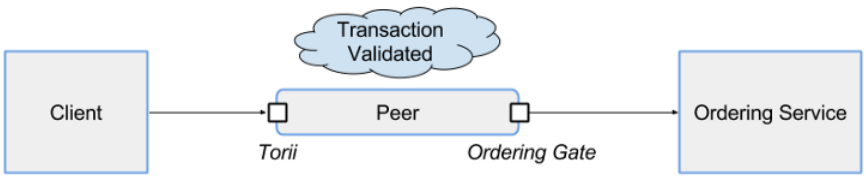

Figure 6.1: Iroha transaction flow

7 Performance benchmark

For benchmarking, the **Hyperledger Caliper**, a blockchain performance testing tool, was used. *Caliper* deploys the tested blockchain networks and its peers locally in separate Docker containers and then uses **RPC** for performing the performance benchmark[60].

7.1 Hyperledger Caliper

Caliper currently collects the following performance indicators[61]:

- **Success rate** - This parameter indicates how many transactions out of all the submitted transactions have been successfully processed and written on the blockchain.

  A failed transaction could be due to the time-outs, wrong network configuration or bugs in the smart contracts.

- **Transaction/Read throughput** - This reports the number of transactions (or queries) per second (TPS) that was processed by the underlying blockchain network.

- **Transaction/Read latency** (minimum, maximum, average) - These metrics are based on the time that takes for a transaction or query from the submission by the client until it is processed and is written on the ledger.

- **Resource consumption** (CPU, Memory, Network IO, ...) - These indicators show the resource that is consumed by each Docker container (each representing a peer in the blockchain).

7.1.1 RPC

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.

RPC uses the *client-server model*. The requesting program is a client and the service providing program is the server.[62].

7.2 Performance benchmark setup

To ensure as consistent results as possible, each of the blockchain networks was set up with the same number of nodes - **two**. The performance benchmarking phase consist of **three** phases:

1. **open** - This is the first phase of the benchmarking that on each node opens and account with the initial funds of **1000 units**.
2. **query** - The second phase of the benchmark where a node sends queries to the network.
3. **transfer** - The last phase, where a node transfers **10 units** of its balance to the another account residing on the other node in the network.

During each phase, the *Caliper* gathers all the performance indicators that are described in section 7.1.

7.2.1 Send rate variations

The performance benchmark was run **four** different times, each time with a different **fixed send rate** (denoted in *transactions per second* - **TPS**). The send rate indicates the number of times per second each step of the benchmark is executed.

The values of the send rate were fixed to **10**, **20**, **50**, and **100** TPS. In section 7.3, the compacted results of each *phase* for each *send rate* are located. These results contain:

- success rate (%)
- throughput (TPS)
- average latency (s)
- average memory consumption (MB)
- average CPU utilization (%)

For full results, refer to section B.2.

Note: For *Iroha*, the performance indicators for the **transfer** phase are omitted as the current version of **Caliper** (v 0.2.0) does not implement this feature yet.

7.3 Performance benchmark results

7.3.1 10 TPS send rate

Table 7.1: Account opening at 10 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	7,5	5,05	629,9	27,62
Fabric	100	9,8	0,28	219,4	22,67
Besu	100	7,1	3,61	289,2	75,61
Iroha	100	9,0	2,37	229,7	32,02

Table 7.2: Account querying at 10 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	10,0	0,04	735,7	33,49
Fabric	100	10,1	0,04	223,7	19,22
Besu	100	10,0	0,04	318,7	59,95
Iroha	100	10,1	0,02	236,0	6,34

Table 7.3: Asset transferring at 10 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	7,2	5,03	736,8	39,30
Fabric	95	9,3	0,28	225,0	33,46
Besu	100	6,7	3,34	322,9	65,99

7.3.2 20 TPS send rate

Table 7.4: Account opening at 20 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	8,3	5,05	641,2	22,86
Fabric	100	19,1	0,32	224,5	23,42
Besu	100	10,3	4,18	256,6	83,26
Iroha	100	8,9	5,92	230,6	14,88

Table 7.5: Account querying at 20 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	20,2	0,02	756,0	42,34
Fabric	100	20,1	0,02	240,5	23,03
Besu	100	20,1	0,02	258,3	82,72
Iroha	100	20,2	0,02	235,5	7,78

Table 7.6: Asset transfering at 20 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	10,1	5,04	756,8	30,05
Fabric	86	16,4	0,32	249,6	52,50
Besu	99	16,5	3,37	267,5	99,21

7.3.3 50 TPS send rate

Table 7.7: Account opening at 50 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	11,4	5,40	651,3	29,57
Fabric	100	25,3	0,62	268,8	17,16
Besu	100	10,3	6,07	221,0	70,75
Iroha	100	8,8	9,51	235,5	9,36

Table 7.8: Account querying at 50 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	50,3	0,02	750,5	37,65
Fabric	100	50,0	0,05	309,6	23,75
Besu	100	50,2	0,05	250,2	78,58
Iroha	100	50,1	0,02	243,0	14,73

Table 7.9: Asset transfering at 50 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	14,2	5,58	751,4	32,79
Fabric	53	15,2	1,13	320,2	40,15
Besu	100	14,3	4,60	256,9	82,82

7.3.4 100 TPS send rate

Table 7.10: Account opening at 100 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	13,1	6,48	808,0	36,60
Fabric	100	25,5	0,56	181,0	21,74
Besu	100	10,0	6,58	266,0	77,34
Iroha	100	9,1	9,06	230,3	32,43

Table 7.11: Account querying at 100 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	93,7	0,06	950,8	5,20
Fabric	100	65,2	0,10	203,1	35,46
Besu	100	68,5	0,56	257,8	0,36
Iroha	100	83,5	0,04	238,0	0,23

Table 7.12: Asset transfering at 100 TPS

Blockchain	Success rate [%]	Throughput [TPS]	Latency [s]	RAM [MB]	CPU [%]
Ethereum	100	16,2	5,07	988,4	43,75
Fabric	51	13,2	1,12	208,3	50,56
Besu	99	12,1	4,32	286,2	66,13

8 Comparison

In this chapter, all four blockchains will be compared head-to-head from two main points of view:

- **Functional comparison** - section 8.1 describes similarities and differences from the functional point of view such as *consensus levels and mechanisms*, *transaction execution*, etc.
- **Performance comparison** - section 8.2 compares the blockchain performance indicators collected running the custom benchmark (section 7.2) against each of the blockchains.

8.1 Functional comparison

When considering a *functional* comparison of blockchains, the aspects that are taken into account are the ones that are shared among all blockchain technologies.

These aspects are[63]:

- Use case
- Mode of operation
- Consensus model
- Transaction model
- Operational governance
- Privacy support
- Smart contract management
- Currency and token support

8.1.1 Use case

Ethereum

Ethereum is general-purpose, independent of any field. Mostly used in P2P[1] and B2C[2] communication.

Hyperledger Fabric

Fabric has a modular, extendable architecture for various industries, mostly used in B2B[3] communication.

Besu

Besu is multi-purpose, designed for any enterprise use case.

Iroha

Iroha's main use cases are: IoT, certificates in education and healthcare, cross-border asset transfer, and financial applications[64].

8.1.2 Mode of operation

Table 8.1: Mode of operation comparison

	Ethereum	Fabric	Besu	Iroha
type	public	private	hybrid	private
permissioned	no	yes	—	yes

With Ethereum's public and permissionless model, any node can join the network without any restrictions.

Hyperledger Fabric and Iroha are private, permissioned which means only pre-authorized nodes are able to join the network.

Besu, as a hybrid blockchain, can be public, permissionless on an Ethereum network but providing private transactions or run on a dedicated test network.

1. Peer-to-Peer
2. Business-to-Customer
3. Business-to-Business

8.1.3 Consensus model

Table 8.2: Consensus model comparison

	Ethereum	Fabric	Besu	Iroha
Model	PoW	*custom*	PoW, Clique, IBFT 2.0	YAC
Level	ledger	transac.	ledger	transac.

Ethereum

In Ethereum, all nodes must first agree on the transaction order inside a block, then, the transactions are independently executed by each node to calculate the resulting state - called the **order-and-execute** consensus model where every transaction execution must be deterministic. This is achieved by removing non-deterministic features such as clocks, floating point arithmetic, etc.

Hyperledger Fabric

Fabric, instead of eliminating, tolerates non-determinism in its consensus model for smart contracts. This is made possible by adopting the unique **execute-order-validate** consensus design, so that the blockchain can endure bad transactions caused by Chaincode.

Besu

The consensus model of Besu depends on its mode of operation. If Besu run on a public network utilizing the *Proof of Work* algorithm, it follows the same **order-and-execute** model as Ethereum. When Besu uses any of the *Proof of Authority* mechanisms, consensus is achieved utilizing pre-elected validators.

Iroha

In the YAC algorithm in Iroha, the *ordering service* first orders the transactions, then sends it to the peers on the network for signing. After signing, the *leader* collects the votes and then commits the transaction to the peers. Lastly, the peers verify the commit and add the block to the ledger.

8.1.4 Transaction model

Ethereum

Ethereum invented the *Account State* transaction model, which got adopted by Fabric and Besu as well. The smart contracts maintain a world state that is scoped to that smart contract and each transaction can update or query the latest values.

When executing a transaction, Ethereum smart contract assumes own identity when calling other contracts which allows the downstream contracts to enforce access controls.

Hyperledger Fabric

Chaincodes, compared to Ethereum smart contracts, are completely free of identities. A chaincode can also call another chaincode, but the transaction identity is always kept constant as the transaction signer who submitted the transaction throughout the calling chain.

Iroha

A transaction in Iroha is straightforward. Users can create and manage their assets via Iroha commands.

8.1.5 Operational governance

Governance is the process and policy around the protocol runtime to ensure consortium member organizations have proper participation in the decision making processes.

Table 8.3: Operational governance

Ethereum	Fabric	Besu	Iroha
Ethereum developers	Linux Foundation		

8.1.6 Privacy support

Table 8.4: Privacy model comparison

Ethereum	Fabric	Besu	Iroha
public	public & private	private	private

Ethereum

On an Ethereum network, every transaction ever made is publicly available for viewing including the sender and recipient address, and the amount.

Hyperledger Fabric

In Fabric, the private state is calculated during the endorsement phase by the peers executing the private transaction. As a result, consistent hash is present on all nodes after the consensus guarantees that the participating nodes of the private transactions agree on the states.

Fabric also offers complete data isolation with the concept of *channels*. Fabric channels can be thought of as separate blockchains, because each channel maintains its completely separate instance of a ledger, shared among the participating nodes of that channel.

Besu

Besu uses a *Private Transaction Manager* to implement privacy. Private transactions are passed from the Besu node to the associated Orion node. The Orion node encrypts and directly distributes (that is, point to point) the private transaction to Orion nodes participating in the transaction.

Iroha

Iroha allows users to perform common functions, such as creating and transferring digital assets, by using pre-built commands that are in the system. It is the only ledger that has a robust permission system, allowing permissions to be set for all commands, queries, and joining of the network.

8.1.7 Smart contract management

Ethereum

Ethereum's smart contracts are immutable and after deployment, they cannot be changed. Upgrading a smart contract in Ethereum requires one of two approaches:

- Upgradable Contracts through a Proxy Contract – where the original contract code is written with migration in mind. So that all of the logic and state is upgradable, without changing the interface or address.
- Contract Migration – deploying a new contract, which gets assigned a new address, and copying over the existing state from the old version.

Hyperledger Fabric

Deploying a Chaincode is a two-step process:

1. Installation - Because the executable binary for the Chaincode does not actually live on the ledger, they must first be installed on every endorsing peer selected to support it.
2. Chaincode instantiation - The whole channel must agree on the exact version of the Chaincode to execute and the endorsement policy for transactions, by a step called *Chaincode instantiation*.

The user submitting the transaction must pass the validation against the *instantiation policy* to ensure they are approved to do so according to the predetermined rules when the consortium was established.

Besu

Besu has a similar mode of smart contract management as Ethereum with tools, such as *Truffle*, with the only difference being Besu does not implement private key management.

Iroha

There are no smart contracts in Iroha.

8.1.8 Currency and token support

Tokens is a form in which some quantity of cryptocurrency is sold. Typical features of a token include the following:

- Ownership - who owns a particular token (non-fungible) or what amount of tokens (fungible).
- Transfer - To allow the digital assets to be traded.
- Minting - For new tokens to be created for circulation.
- Burning - Owners are allowed to *burn* tokens.

  This is used in cross-chain trades where they get burned in the sender's chain and minted in the receiver's chain.
- Custody - Owners can approve of accounts to look after a portion of their tokens.

  In scenarios where an escrow is involved, the escrow account can be approved as a custodian by the tokens owner.

Table 8.5: Currency and token support

	Ethereum	Fabric	Besu	Iroha
Native cryptocurrency	*Ether*	none	*Ether* support	none
Tokens	via smart contract	via Chain-code	via smart contract	

8.2 Performance comparison

This section contains **line graphs** of some of the performance results of the **open** and the **transfer** phase.

8.2.1 Account opening graphs

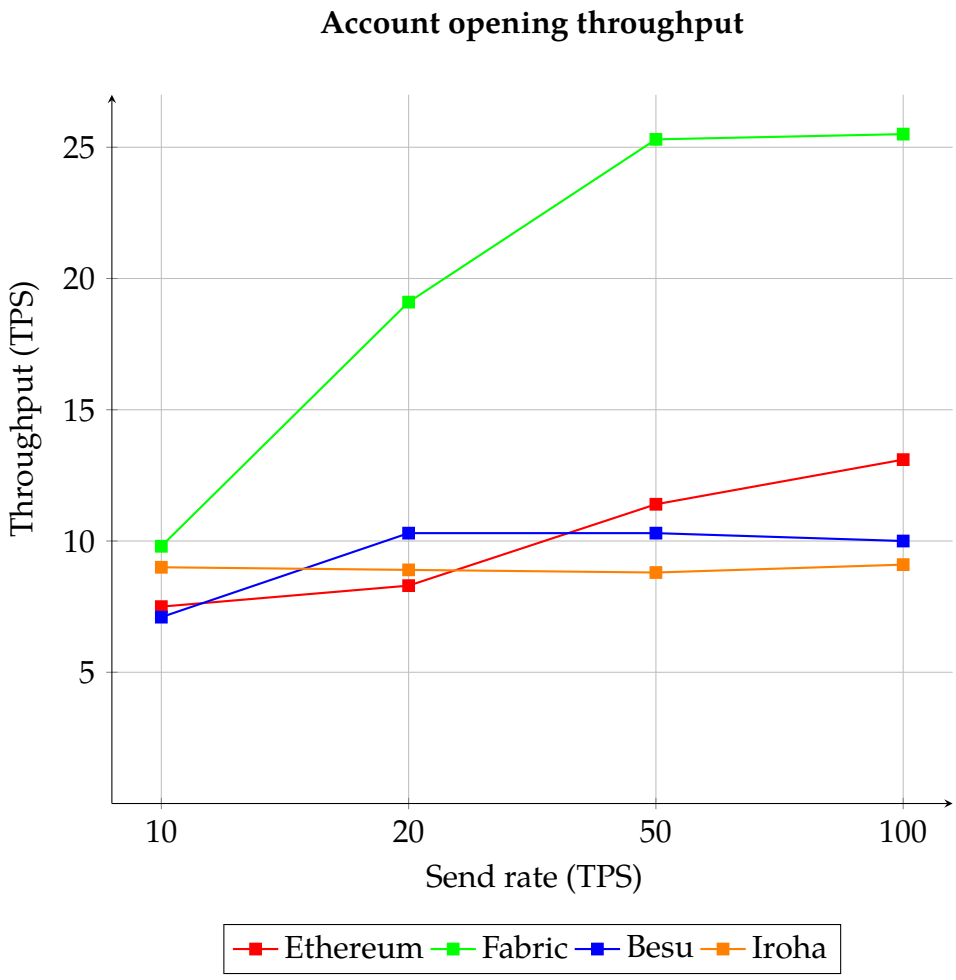

Figure 8.1: Throughput comparison of account opening

Figure 8.1 shows that *Fabric* outperformed all the other blockchains with increasing *send rate* by more than 12 transactions at 100 TPS.

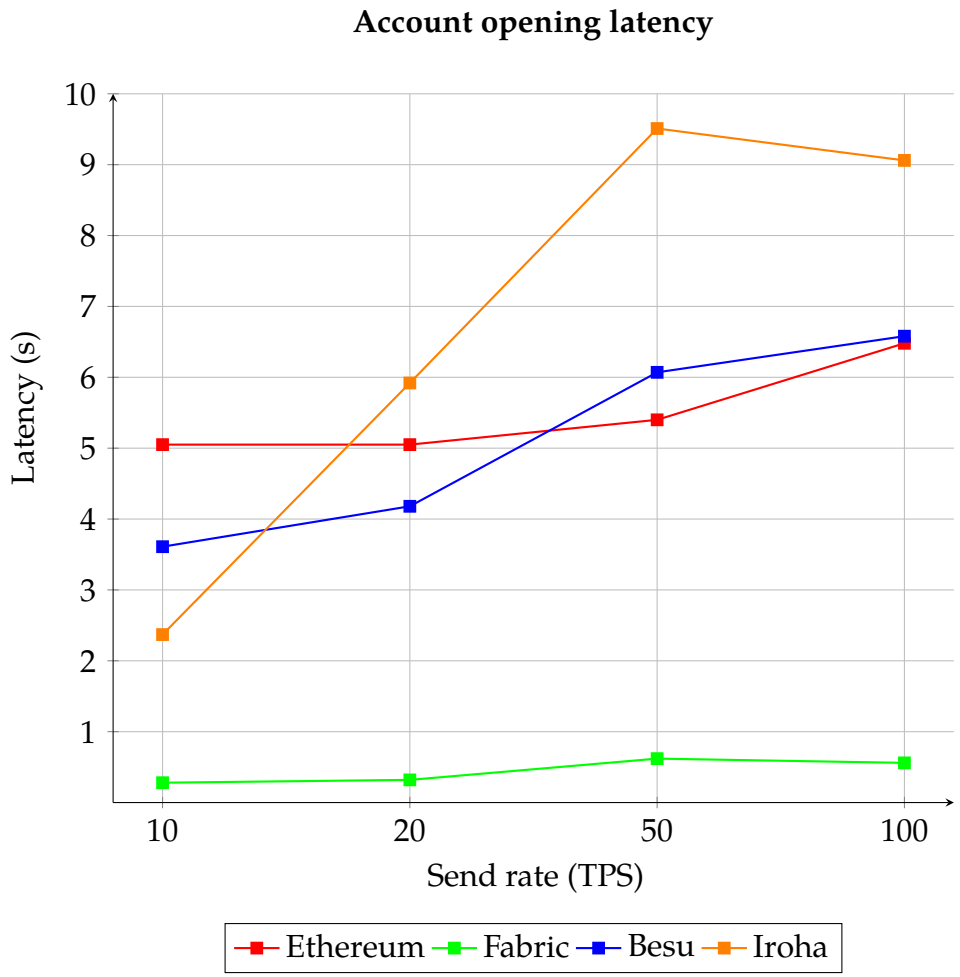

Figure 8.2: Latency comparison of account opening

Figure 8.2, which compares the average account opening latency shows that *Fabric* had to lowest *latency* of all the tested blockchain across all send rates by a significant margin. The latency was below 1 second for all send rates.

While *Iroha* had better latency at 10 TPS than Ethereum or Besu, at higher send rates, the latency showed worse performance than the slightly increasing indicators of Ethereum and Besu.

Ethereum's latency was lower than Besu's for 10 TPS and 20 TPS but with higher send rates, the latency increased and for faster send rates Besu performed better.

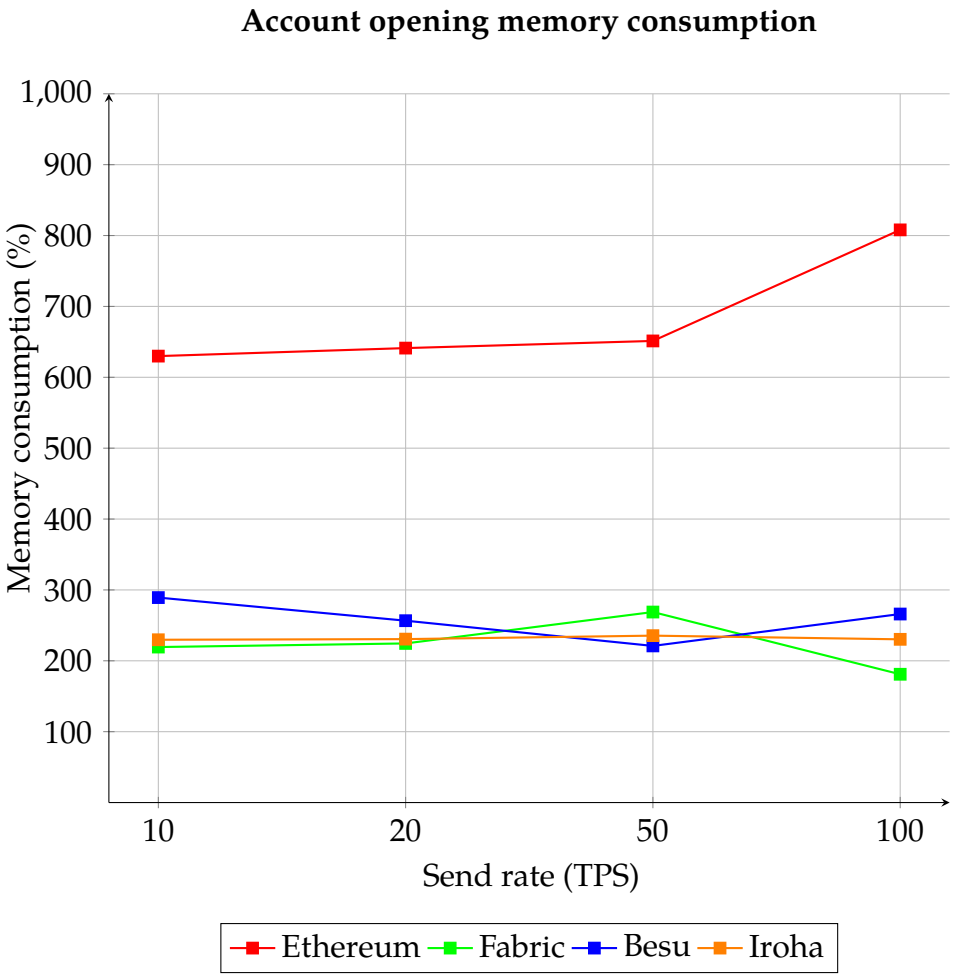

Figure 8.3: Memory consumption comparison of account opening

Figure 8.3 compares the average memory consumption of all the blockchain networks.

Ethereum was at all times at least twice worse with the memory consumption than any other blockchain.

With increased send rate, Fabric, Besu, and Iroha had similar memory consumption that varies between 180 MB - 300 MB while Ethereum's memory consumption slightly increased with a bigger jump between 50 TPS and 100 TPS send rate.

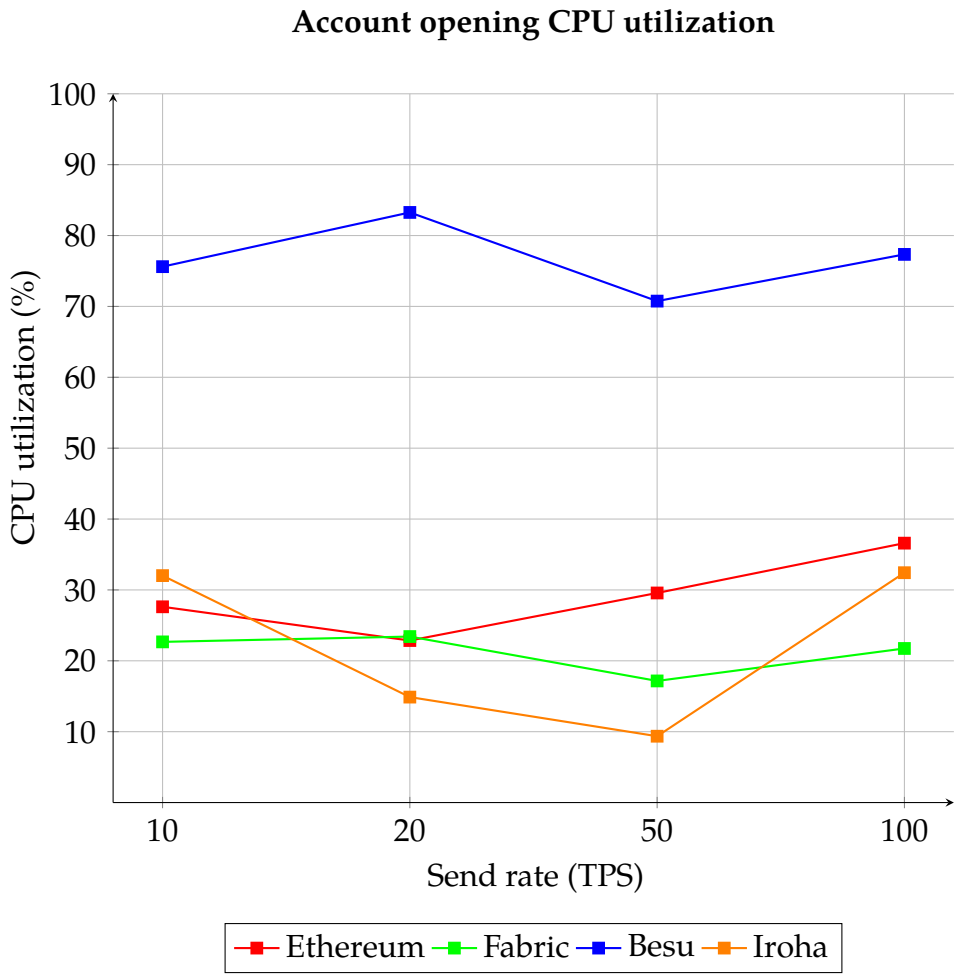

Figure 8.4: CPU utilization comparison of account opening

Figure 8.4 shows CPU utilization for each send rate of all the tested blockchain networks.

In this regard, Besu used by far the most ticks of the CPU with values between 70 % and 85 % for all send rates.

All other blockchains used less than 40 % of the CPU power where Ethereum varied between 20% and 40% and had a slight and steady increase after 20 TPS send rate.

Fabric constantly performed at around 20 % while Iroha varied between approximately 10 % and 35 %.

8.2.2 Asset transfer graphs

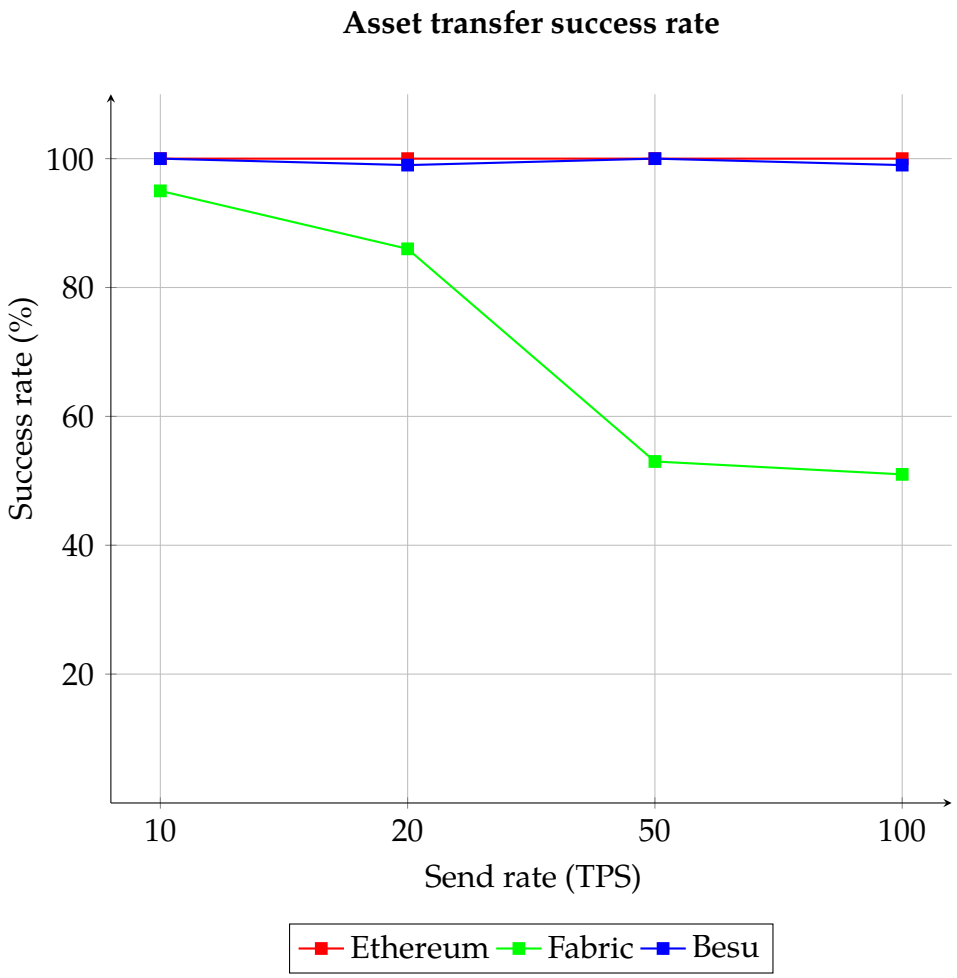

Figure 8.5: Success rate comparison of asset transfer

Figure 8.5 compares the blockchains with the regard to their transfer success rate.

Ethereum had a success rate 100% for all the send rates while Besu fell behind by only 1% at 20 TPS and 100 TPS.

Fabric had never reached 100% success rate with the highest being 95% at 10 TPS falling down with each increasing send rate. At 50 TPS, Fabric had 53% and at 100 TPS only had 51% success rate.

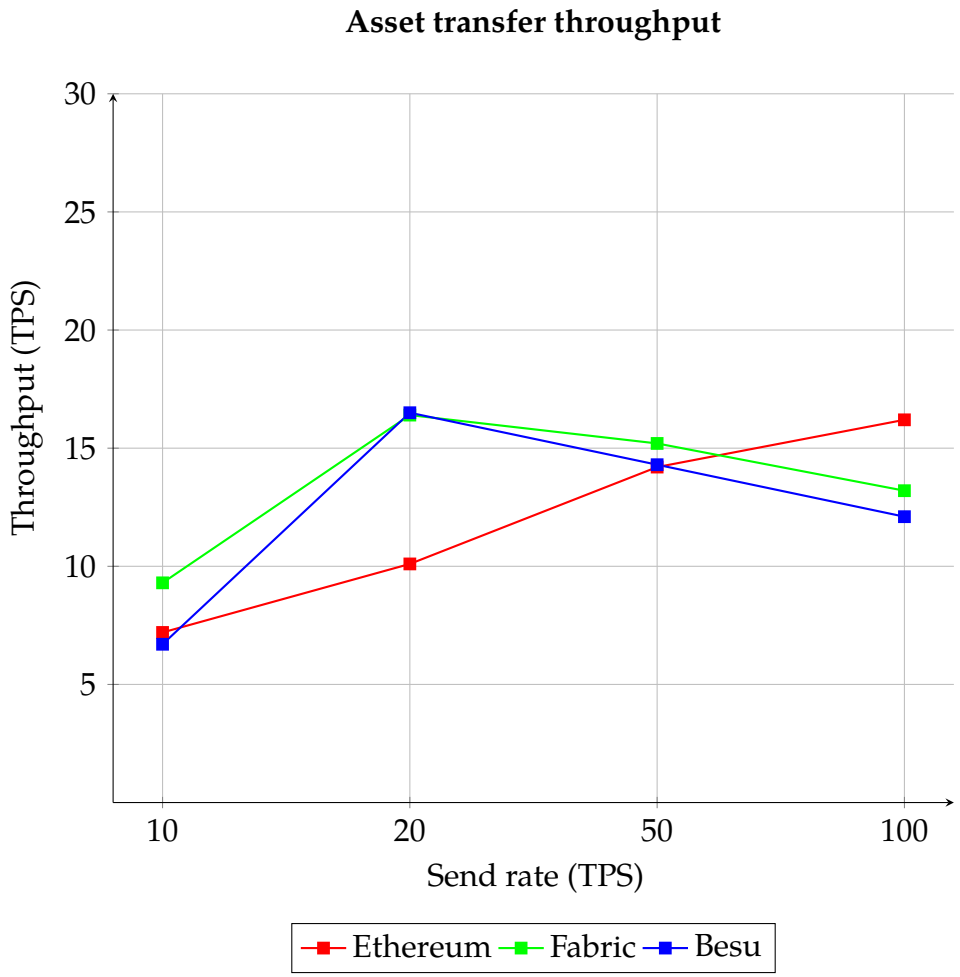

Figure 8.6: Throughput comparison of asset transfer

Figure 8.6 compares the blockchains with regard to their throughput while tranferring assets.

Ethereum, with each increased send rate, increased the transactions per second in can handle. The other blockchains peeked at around 17 transactions per second at 20 TPS send rate while dropping down on performance with higher send rate.

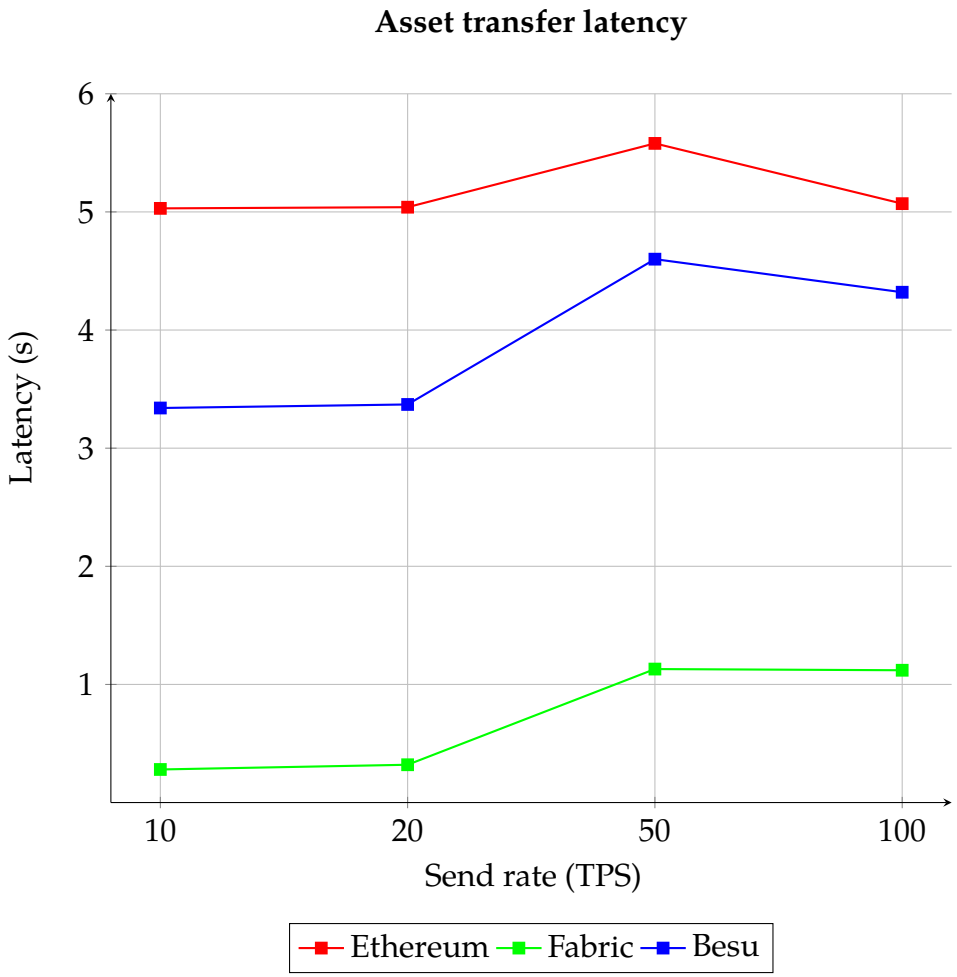

Figure 8.7: Latency comparison of asset transfer

Figure 8.7 compares the blockchain networks by their latency while transferring assets.

All the blockchains showed the same tendency by having almost equal values for their respective 10 TPS and 20 TPS send rate, peeking at 50 TPS send rate and then falling back down a little.

Ethereum showed the biggest latency between 5 seconds and 6 seconds. Besu was slightly faster, varying between 3 seconds and 4,5 seconds while Fabric again outperformed the other blockchains significantly with latency between a few milliseconds to 1,2 seconds.

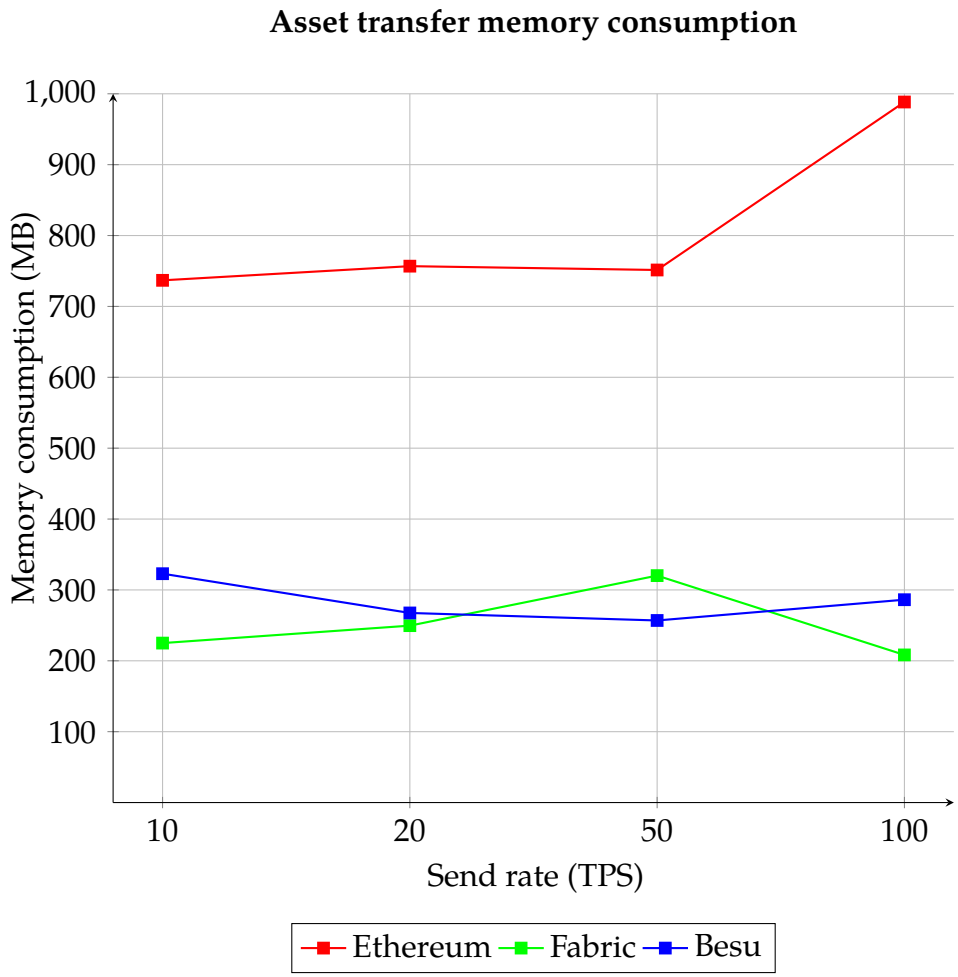

Figure 8.8: Memory consumption comparison of asset transfer

Figure 8.8 shows a head-to-head comparison of the memory usage for executing a transaction.

In this case, Ethereum also consumed much more memory as it was in the case of account opening. Ethereum and Besu varied between 200 MB - 350 MB with only small changes between different send rates.

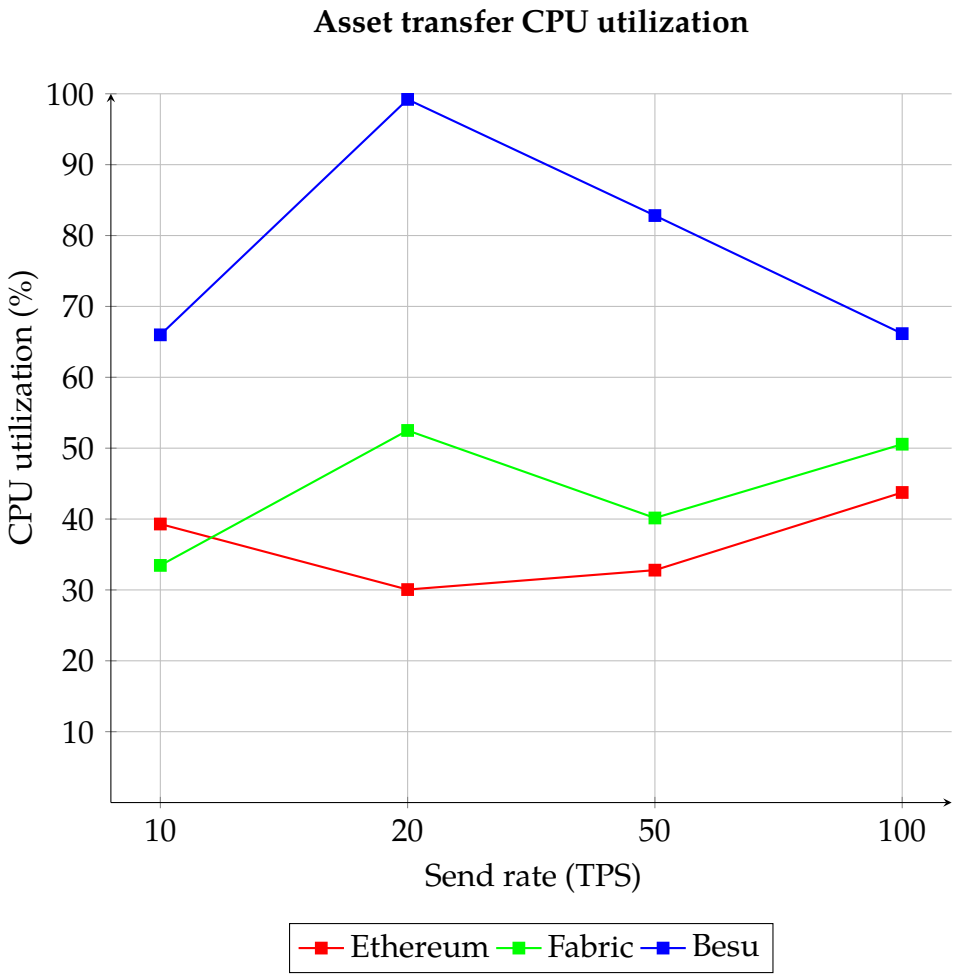

Figure 8.9: CPU utilization comparison of asset transfer

Figure 8.9 shows comparison of CPU utilization of asset transfer for all tested blockchain networks.

When it comes to CPU utilization, Besu used the most CPU ticks as it was the case while opening an account. Ethereum and Fabric showed similar values, except for the 20 TPS send rate when Fabric performed much worse.

8.2.3 Comprehensive performance comparison

The performance benchmark from section 7.2 that was carried out on the blockchains resulted in performance indicators that are applicable only to this benchmark setup.

Note that running the attached benchmark on different workstations might result in different numbers as the performance is partly given by the power of the underlying hardware while taking into account what other processes than the benchmark use the workstation's resources.

The performance, in general, may vary based on the following factors[65][66]:

1. Network's send rate
2. Smart contract implementation
3. Size of the transactions
4. Consensus algorithm
5. Network size
6. Hardware capabilities of the nodes

Based on the results from section 7.3 and section B.2, the following conclusions can be drawn:

Success rate

All the tested blockchains had a 100% success rate in **open** and **query** phases.

For **transfer**, Fabric had the worst success rate of all while there was no distinct difference between the other blockchains.

Throughput

At the **open** phase, Iroha had the worst performance regardless of the send rate. Ethereum and Besu are very similar in their throughput. The best performer at the produced throughput was Fabric.

The **query** showed almost every time the maximum possible performance for all blockchains.

The benchmark showed only minor differences during **transfer**. Ethereum is currently limited to around 15 TPS but this will change when Ethereum changes its consensus algorithm from Proof of Work to Proof of Stake[67]. With a different setup, Fabric might reach 3500 TPS[68].

Latency

Fabric, has the lowest average latency for almost every case when considering **open** and **transfer** phases. In these phases Iroha performs the worst while there are no significant differences between Ethereum and Besu.

There are negligible differences while performing a network **query**.

Resource consumption

For all testing phases, Ethereum has the highest RAM consumption. There are smaller, insignificant differences between the other tested blockchains.

Also for all testing phases, Besu has the most CPU utilization while Iroha has the least, often only $\frac{1}{5}$ of Besu.

Data transfer

For all intents and purposes, Fabric utilizes tho most of network traffic but it did not top a couple of megabytes.

The other blockchains use only a few hundreds of kilobytes of network traffic.

Disc read and write operations are mostly non-existent for all blockchains.

8.3 Advantages and disadvantages

Almost any of the blockchain's properties can be its advantage or disadvantage. The difference in the **use case** for which a blockchain is considered to be used.

8.3.1 Ethereum

The main advantage of Ethereum is its wide usage and active developer community. Another advantage is the reward a miner receives for expending computational power during mining.

The disadvantages of Ethereum include: difficult smart contract upgrade, only Proof of Work as a consensus mechanism and high resource usage.

8.3.2 Fabric

Fabric's greatest advantage is its speed compared to the other blockchains. Fabric also support more consensus mechanisms, different ordering services, and potentially a higher throughput.

The main downside o Fabric is a decreasing success rate with the increased send rate.

8.3.3 Besu

The main strength of Besu is the different types consensus algorithms it can use. It also has an advantage of supporting private transactions on a public channel. Besu also provides a granular permission model.

All these advantages, however, come with a large performance overhead, particularly CPU utilization.

8.3.4 Iroha

Iroha's biggest strength is its simplicity which makes it suitable for IoT where nodes typically come with hardware having less performance.

The disadvantage of Iroha is only YAC as its consensus mechanism.

9 Conclusion

The aim of this thesis was to describe the blockchain technology, in-detail describe at least three open-source implementations of the blockchain, run a defined test case, collect and analyze statistical data collected during testing, compare the selected blockchains and discuss their advantages and disadvantages.

The second chapter introduced blockchain, the fundamentals and data structures blockchain is built on, consensus mechanisms, attacks on blockchain networks, challenges and metrics used while testing blockchain networks.

The third chapter described Ethereum, EVM, gas, and the transaction execution with Ethereum's Proof of Work consensus mechanism.

In the fourth chapter, Hyperledger Fabric was presented. This chapter also described the channels, types of peers, and the transaction flow with the ordering services in Fabric.

The fifth chapter presented Hyperledger Besu with the variety of its consensus mechanisms and permissioning model.

The sixth chapter described the Hyperledger Iroha blockchain together with its transaction execution and its unique Yet another consensus algorithm.

The seventh chapter introduced the test cases used to performance benchmark the blockchains together with results for each test run.

The eight chapter then compared the blockchains from their functional and performance point of view using line charts.

The results of the comparison, based on the eight chapter, relative to each blockchain's use case showed that Ethereum's strong properties are its community and miner reward whereas its weak points are difficult scalability, limited consensus mechanisms and higher resource consumption. Fabric's advantages are mainly the low latency, higher throughput and customizable consensus mechanisms while its main disadvantage is a lower success rate. Besu's strong point include private transactions over public channels, different types of consensus mechanisms and granular permission model met, on the other hand, with higher CPU utilization. Iroha's main advantage is its simplicity which is suitable for hardware with lesser performance with the disadvantage being only one consensus mechanism.

Bibliography

1. CROSBY, Michael; PATTANAYAK, Pradan; VERMA, Sanjeev; KALYANARAMAN, Vignesh, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*. 2016, vol. 2, no. 6-10, pp. 71.
2. WOOD, Gavin. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER* [online]. The "Yellow Paper": Ethereum's formal specification [visited on 2019-10-06]. Available from: `https://ethereum.github.io/yellowpaper/paper.pdf`.
3. NARAYANAN, Arvind; BONNEAU, Joseph; FELTEN, Edward; MILLER, Andrew; GOLDFEDER, Steven. *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press, 2016.
4. BECKER, Georg. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*. 2008. Ruhr-Universität Bochum.
5. *Hash tree* [online]. Wikimedia Commons [visited on 2019-04-14]. Available from: `https://en.wikipedia.org/wiki/Merkle_tree#/media/File:Hash_Tree.svg`.
6. BLOOM, Burton Howard. Space/time trade-offs in hash coding with allowable errors. 1970, vol. 13, no. 7. Available from DOI: `10.1145/362686.362692`.
7. MITZENMACHER, Michael; UPFAL, Eli. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005. ISBN 9780521835404.
8. *The Truth About Blockchain* [online]. Marco Iansiti and Karim R. Lakhani [visited on 2019-11-17]. Available from: `https://hbr.org/2017/01/the-truth-about-blockchain`.
9. *What is Decentralization?* [online]. Lisk, 2016–2009 [visited on 2019-05-08]. Available from: `https://lisk.io/academy/blockchain-basics/benefits-of-blockchain/what-is-decentralization`.

10. *Blockchain Transparency Explained* [online]. Lisk, 2016–2009 [visited on 2019-05-08]. Available from: https://lisk.io/academy/blockchain-basics/benefits-of-blockchain/blockchain-transparency-explained.

11. *What is Blockchain Technology? A Step-by-Step Guide For Beginners* [online]. Blockgeeks, 2016–2009 [visited on 2019-05-08]. Available from: https://blockgeeks.com/guides/what-is-blockchain-technology/.

12. *21 Promising Blockchain Use Cases* [online]. Mycryptopedia [visited on 2019-06-02]. Available from: https://www.mycryptopedia.com/16-promising-blockchain-use-cases/.

13. KRISHNAN, Hari; SAKETH, Sai; TEJ, Venkata. Cryptocurrency Mining – Transition to Cloud. 2015, vol. 6, no. 9. ISSN 2156-5570. Available from DOI: 10.14569/IJACSA.2015.060915.

14. TAPSCOTT, Don; TAPSCOTT, Alex. *The Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. Portfolio, 2016. ISBN 978-0670069972.

15. *How Do Ethereum Smart Contracts Work?* [online]. Alyssa Hertig [visited on 2019-10-13]. Available from: https://www.coindesk.com/information/ethereum-smart-contracts-work.

16. LAMPORT, Leslie; ROBERT, Shostak; PEASE, Marshall. *The Byzantine Generals Problem* [online]. State College, Pennsylvania, 1982 [visited on 2019-06-02]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.9525&rep=rep1&type=pdf. Pennsylvania State University.

17. DWORK, Cynthia; NAOR, Moni. *Pricing via Processing or Combatting Junk Mail*. CRYPTO 1992: Advances in Cryptology — CRYPTO' 92. 1992, vol. 740. ISBN 978-3-540-48071-6. Available from DOI: 10.1007/3-540-48071-4_10.

18. *Proof of work* [online]. Bitcoin Wiki [visited on 2019-06-02]. Available from: https://en.bitcoin.it/wiki/Proof_of_work.

19. *Understanding Blockchain Fundamentals, Part 2: Proof of Work & Proof of Stake* [online]. Medium [visited on 2019-06-02]. Available from: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-2-proof-of-work-proof-of-stake-b6ae907c7edb.

20. DZIEMBOWSKI, Stefan; FAUST, Sebastian; KOLMOGOROV, Vladimir; PIETRZAK, Krzysztof. *Proofs of Space* [Cryptology ePrint Archive, Report 2013/796]. 2013. https://eprint.iacr.org/2013/796.

21. *6 green cryptocurrencies to watch: How to choose eco-friendly digital tokens* [online]. DSX [visited on 2019-10-13]. Available from: https://dsx.uk/blog/6-green-cryptocurrencies-to-watch-how-to-choose-eco-friendly-digital-tokens.

22. *Proof of Authority Explained* [online]. John Ma [visited on 2019-10-13]. Available from: https://www.binance.vision/blockchain/proof-of-authority-explained.

23. *51% Attack* [online]. Investopedia [visited on 2019-10-06]. Available from: https://www.investopedia.com/terms/1/51-attack.asp.

24. *Blockchain: how a 51% attack works (double spend attack)* [online]. Medium [visited on 2019-10-12]. Available from: https://medium.com/coinmonks/what-is-a-51-attack-or-double-spend-attack-aa108db63474.

25. *Sybil Attack* [online]. Geeks for Geeks [visited on 2019-10-20]. Available from: https://www.geeksforgeeks.org/sybil-attack/.

26. *The Best Open Source Blockchain Platforms* [online]. Vivek Ratan [visited on 2019-10-13]. Available from: https://opensourceforu.com/2019/08/the-best-open-source-blockchain-platforms/.

27. WALKER, Martin. *Front-to-Back: Designing and Changing Trade Processing Infrastructure*. Risk Books, 2018. ISBN 978-1-78272-389-9.

28. *Top Blockchain Platforms to watch out in 2019* [online]. Hackernoon [visited on 2019-10-06]. Available from: https://hackernoon.com/top-blockchain-platforms-to-watch-out-in-2019-aa80e336a426.

29. *TOP 10 BLOCKCHAIN PLATFORMS YOU NEED TO KNOW ABOUT* [online]. Blockchain-council [visited on 2019-10-06]. Available from: https://www.blockchain-council.org/blockchain/top-10-blockchain-platforms-you-need-to-know-about/.

30. *TOP 10 BLOCKCHAIN PLATFORMS YOU NEED TO KNOW ABOUT* [online]. Arpatech [visited on 2019-10-13]. Available from: https://www.arpatech.com/blog/top-trending-open-source-blockchain-platforms/.

31. *Blockchain Application Testing* [online]. TestingXperts [visited on 2019-10-16]. Available from: https://www.testingxperts.com/services/blockchain-application-testing/.

32. *BLOCKCHAIN TESTING* [online]. Smartsourcing [visited on 2019-10-16]. Available from: http://www.smartsourcingglobal.com/blockchain-testing-2/.

33. LEWIS, Antony. *The Basics of Bitcoins and Blockchains: An Introduction to Cryptocurrencies and the Technology that Powers Them*. Mango Publishing, 2018. ISBN 1633538001.

34. *Ethereum Explained: Merkle Trees, World State, Transactions, and More* [online]. Gina Rubino [visited on 2019-10-13]. Available from: https://pegasys.tech/ethereum-explained-merkle-trees-world-state-transactions-and-more/.

35. *What is Gas?* [online]. MyEtherWallet [visited on 2019-10-13]. Available from: https://kb.myetherwallet.com/en/transactions/what-is-gas/.

36. *Ethereum 101 - Part 4 - Accounts, Transactions, and Messages* [online]. Wil Barnes [visited on 2019-10-14]. Available from: https://kauri.io/article/7e79b6932f8a41a4bcbbd194fd2fcc3a/ethereum-101-part-4-accounts-transactions-and-messages.

37. *Introduction to Blockchain, Ethereum and Smart Contracts — Chapter 1* [online]. Ritesh Modi [visited on 2019-10-14]. Available from: https://medium.com/coinmonks/https-medium-com-ritesh-modi-solidity-chapter1-63dfaff08a11.

38. *How does Ethereum work, anyway?* [online]. Medium [visited on 2019-10-06]. Available from: https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369.

39. *Life Cycle of an Ethereum Transaction* [online]. Mahesh Murthy [visited on 2019-10-14]. Available from: https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e.

40. *The Ethereum Virtual Machine — How does it work?* [online]. Luit Hollander [visited on 2019-10-14]. Available from: https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e.

41. *Ethash Explained* [online]. Bisade Asolo [visited on 2019-10-14]. Available from: https://www.mycryptopedia.com/ethash-explained/.

42. *Hyperledger Fabric* [online]. The Linux Foundation [visited on 2019-10-09]. Available from: https://www.hyperledger.org/projects/fabric.

43. *Hyperledger Fabric – Are Channels Private Blockchain? (Deep Dive)* [online]. Ajitesh Kumar [visited on 2019-10-15]. Available from: https://vitalflux.com/hyperledger-fabric-channels-private-blockchain-deep-dive/.

44. *Endorsement Policies in Hyperledger Fabric* [online]. Swapnil Kulkarni [visited on 2019-10-16]. Available from: https://opensourceforu.com/2019/07/endorsement-policies-in-hyperledger-fabric/.

45. *How does Hyperledger Fabric work?* [online]. Muntasir Mamun [visited on 2019-10-15]. Available from: https://medium.com/coinmonks/how-does-hyperledger-fabric-works-cdb68e6066f5.

46. *Demystifying Hyperledger Fabric ordering and decentralization* [online]. Arnaud Le Hors [visited on 2019-10-16]. Available from: https://developer.ibm.com/articles/blockchain-hyperledger-fabric-ordering-decentralization/.

47. *The Ordering Service* [online]. Hyperledger [visited on 2019-10-09]. Available from: https://hyperledger-fabric.readthedocs.io/en/release-1.4/orderer/ordering_service.html.

48. *The ABCs of Kafka in Hyperledger Fabric* [online]. Codeburst [visited on 2019-10-09]. Available from: https://codeburst.io/the-abcs-of-kafka-in-hyperledger-fabric-81e6dc18da56.

49. *Demystifying Hyperledger Fabric ordering and decentralization* [online]. Arnaud Le Hors [visited on 2019-10-16]. Available from: https://developer.ibm.com/articles/blockchain-hyperledger-fabric-ordering-decentralization/.

50. *HYPERLEDGER BESU: AN EXHAUSTIVE GUIDE* [online]. Toshendra Kumar Sharma [visited on 2019-12-05]. Available from: https://www.blockchain-council.org/blockchain/hyperledger-besu-an-exhaustive-guide/.

51. *Hyperledger Besu – The Open Source Hyperledger Public Blockchain* [online]. Nitish Singh [visited on 2019-12-05]. Available from: https://101blockchains.com/hyperledger-besu/.

52. *Consensus Protocols* [online]. Hyperledger [visited on 2019-12-05]. Available from: https://besu.hyperledger.org/en/stable/Concepts/Consensus-Protocols/Overview-Consensus/.

53. *Comparing Proof of Authority Consensus Protocols* [online]. Hyperledger [visited on 2019-12-05]. Available from: https://besu.hyperledger.org/en/stable/Concepts/Consensus-Protocols/Comparing-PoA/.

54. *Scaling Consensus for Enterprise: Explaining the IBFT Algorithm* [online]. PegaSys [visited on 2019-12-05]. Available from: https://media.consensys.net/scaling-consensus-for-enterprise-explaining-the-ibft-algorithm-ba86182ea668.

55. *Hyperledger Besu* [online]. Silona Bonewald [visited on 2019-12-05]. Available from: https://wiki.hyperledger.org/display/BESU/Hyperledger+Besu.

56. *Validating Transactions* [online]. Hyperledger [visited on 2019-12-05]. Available from: https://besu.hyperledger.org/en/stable/Concepts/Transactions/Transaction-Validation/.

57. *Hyperledger IROHA, a Permission-Based Blockchain Project* [online]. Serkan Erkan [visited on 2019-12-05]. Available from: https://medium.com/@serkanerkan/hyperledger-iroha-a-permission-based-blockchain-project-be86546c7d61.

58. MURATOV, Fedor; LEBEDEV, Andrei; IUSHKEVICH, Nikolai; NASRULIN, Bulat; TAKEMIYA, Makoto. *YAC: BFT Consensus Algorithm for Blockchain*. 2018. Available also from: https://arxiv.org/pdf/1809.00554.pdf.

59. *Hyperledger Iroha - Architecture, Functional/Logical Flow & Consensus(YAC) Mechanism* [online]. Chandrasekaran Palanivel [visited on 2019-12-05]. Available from: https://www.linkedin.com/pulse/hyperledger-iroha-architecture-functionallogical-chandrasekaran.

60. *Tool to Measure Blockchain Performance: Hyperledger Caliper* [online]. Medium.com [visited on 2019-11-07]. Available from: https://medium.com/coinmonks/tool-to-measure-blockchain-performance-hyperledger-caliper-f192adfba52.

61. *Hyperledger Caliper Explained and Installation Guide (Ubuntu)* [online]. Nima Afraz [visited on 2019-12-04]. Available from: https://medium.com/@nima.afraz/hyperledger-caliper-explained-and-installation-guide-ubuntu-c38dc16d3dcf.

62. *Remote Procedure Call (RPC)* [online]. Margaret Rouse [visited on 2019-11-07]. Available from: https://searchapparchitecture.techtarget.com/definition/Remote-Procedure-Call-RPC.

63. *Enterprise Blockchain Protocols: A Technical Analysis of Ethereum vs Fabric vs Corda* [online]. Jim Zhang [visited on 2019-11-19]. Available from: https://kaleido.io/a-technical-analysis-of-ethereum-vs-fabric-vs-corda/.

64. *9.3. Use Case Scenarios* [online]. Hyperledger [visited on 2019-12-05]. Available from: https://iroha.readthedocs.io/en/latest/develop/cases.html.

65. *Innovative Solutions For Common Blockchain Performance Issues* [online]. Vikram Khajuria [visited on 2019-12-04]. Available from: https://www.mobileappdaily.com/solutions-for-solving-blockchain-performance-issues.

66. *Blockchain performance issues and limitations* [online]. Sergey Prilutskiy [visited on 2019-12-04]. Available from: https://hackernoon.com/blockchain-performance-issues-and-limitations-78qss3co5.

67. *Who Scales It Best? Blockchains' TPS Analysis* [online]. Taygun Dogan [visited on 2019-12-04]. Available from: https://hackernoon.com/who-scales-it-best-blockchains-tps-analysis-pv39g25mg.

68. *Behind the Architecture of Hyperledger Fabric* [online]. IBM Research Editorial Staff [visited on 2019-12-04]. Available from: https://www.ibm.com/blogs/research/2018/02/architecture-hyperledger-fabric/.

Index

A Running the benchmarks

To successfully run the benchmarks on a local workstation, the following criteria have to be met:

1. Installed `npm`, `docker`, and `docker-compose`.
2. Globally installed `npx` package.
3. The current user is a member of the *docker* group.

A.1 Fulfilling the prerequisites

If any of the previous criteria is not met, consult the following subsections for steps and commands needed to fulfill the prerequisites.

A.1.1 Installing NPM, Docker, or Docker-compose

```
sudo apt-get install npm docker docker-compose
```

A.1.2 Installing the `npx` package globally

```
sudo npm install npx -g
```

A.1.3 Adding the current user to the *docker* group

1. Add the *docker* group (if not exists).

   ```
   sudo groupadd docker
   ```

2. Grant the current user privileges to run *docker* without `sudo`.

   ```
   sudo usermod -aG docker $USER
   ```

3. Log out and log back in to re-evaluate the group membership or run the following command:

   ```
   newgrp docker
   ```

4. *(optional)* Verify that *docker* can run without `sudo`.

   ```
   docker run hello-world
   ```

A.2 Hyperledger Caliper

For running the performance benchmarks, take the following steps:

1. Using the `terminal`, navigate into the *caliper* folder after unzipping the package.
2. Install NPM packages:

   ```
   npm install
   ```

3. Bind the desired blockchain to the SDK (use Table A.1 to supply the values for *blockchain* and *version* parameters):

   ```
   npx caliper bind
   -caliper-bind-sut <blockchain>
   -caliper-bind-sdk <version>
   ```

4. Run the benchmark (use table Table A.1 to supply the value for *path_to_network* parameter):

   ```
   npx caliper benchmark run
   -caliper-workspace .
   -caliper-benchconfig benchmark/config.yaml
   -caliper-networkconfig <path_to_network>
   ```

   Table A.1: Parameter values

blockchain	version	path_to_network
fabric	1.4.0	`fabric/fabric.yaml`
ethereum	1.2.1	`ethereum/ethereum.json`
besu	1.3.2	`besu/besu.json`
iroha	0.6.3	`iroha/iroha.json`

5. The results of the performance benchmarks will be saved to the `caliper` folder in *caliper.log* and *report.html* files.

B Performance benchmark results

This chapter contains the complete performance benchmark results for all the tested blockchains. These results are contained in the *electronic attachment* in the `caliper/results` folder.

The following sections are grouped by the **send rate** constraint and use the following pattern:

- *XX* TPS send rate results - This section contains all collected performance indicators where *XX* denotes the **send rate** of the given benchmark.
  - *Blockchain* - This is the corresponding section to each blockchain.
    - Summary results - Table containing the *success rate* and *throughput*.
    - latency - Table containing the maximal, minimal and average *latency*.
    - Resource consumption - Table containing the maximal and average *memory consumption* and *CPU utilization*.
    - Data transfer - Table containing the total amount of data *transferred* between the nodes on the network and the amount of data *written* or *read* from the disc.

B.1 Benchmark configuration file

The configuration file for the benchmark defined in section 7.2 (at `caliper/benchmark/config.yaml`) contains the benchmark parameters: the **send rate** for each phase, the amount of assets to use as the initial account balance, and the amount to transfer.

Follow the instructions in the file annotations to customize the benchmark configuration.

B.2 Full results

This section contains the full performance benchmark results from the test runs using aggregate values from multiple nodes in the tested blockchain networks.

B.2.1 10 TPS send rate results

Ethereum results at 10 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Ethereum/10 TPS/report.html`.

Table B.1: Ethereum summary results for 10 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	10,1 TPS	7,5 TPS
query	100	0	10,1 TPS	10,0 TPS
transfer	100	0	10,1 TPS	7,2 TPS

Table B.2: Ethereum latencies for 10 TPS

Name	Max latency	Min latency	Avg latency
open	7,10 s	2,10 s	5,05 s
query	0,06 s	0,01 s	0,04 s
transfer	7,12 s	2,06 s	5,03 s

Table B.3: Ethereum resource consumption for 10 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	716,1 MB	629,9 MB	78,71 %	27,62 %
query	735,9 MB	735,7 MB	69,20 %	33,49 %
transfer	738,9 MB	736,8 MB	88,75 %	39,30 %

Table B.4: Ethereum data transfer for 10 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	471,2 KB	285,5 KB	0 KB	0 KB
query	136,9 KB	49,6 KB	0 KB	0 KB
transfer	493,0 KB	253,4 KB	0 KB	0 KB

Fabric results at 10 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Fabric/10 TPS/report.html`.

Table B.5: Fabric summary results for 10 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	10,1 TPS	9,8 TPS
query	100	0	10,1 TPS	10,1 TPS
transfer	95	5	10,1 TPS	9,3 TPS

Table B.6: Fabric latencies for 10 TPS

Name	Max latency	Min latency	Avg latency
open	0,50 s	0,08 s	0,28 s
query	0,10 s	0,01 s	0,04 s
transfer	0,52 s	0,09 s	0,28 s

Table B.7: Fabric resource consumption for 10 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	223,2 MB	219,4 MB	51,03 %	22,67 %
query	223,9 MB	223,7 MB	25,05 %	19,22 %
transfer	226,4 MB	225,0 MB	44,18 %	33,46 %

Table B.8: Fabric data transfer for 10 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	1 848,6 KB	1 626,6 KB	332,0 KB	2 568,0 KB
query	603,7 KB	576,6 KB	0 KB	0 KB
transfer	2 002,6 KB	1 664,3 KB	4,0 KB	2 580,0 KB

Besu results at 10 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Besu/10 TPS/report.html`.

Table B.9: Fabric summary results for 10 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	10,1 TPS	7,1 TPS
query	100	0	10,1 TPS	10,0 TPS
transfer	100	0	10,1 TPS	6,7 TPS

Table B.10: Fabric latencies for 10 TPS

Name	Max latency	Min latency	Avg latency
open	6,43 s	1,06 s	3,61 s
query	0,07 s	0,02 s	0,04 s
transfer	6,14 s	1,04 s	3,34 s

Table B.11: Fabric resource consumption for 10 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	295,7 MB	289,2 MB	160,00 %	75,61 %
query	319,5 MB	318,7 MB	77,12 %	59,95 %
transfer	330,1 MB	322,9 MB	142,69 %	65,99 %

Table B.12: Fabric data transfer for 10 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	432,2 KB	248,1 KB	128,0 KB	0 KB
query	137,4 KB	45,0 KB	0 KB	0 KB
transfer	471,7 KB	262,5 KB	0 KB	0 KB

Iroha results at 10 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Iroha/10 TPS/report.html`.

Table B.13: Fabric summary results for 10 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	10,3 TPS	9,0 TPS
query	100	0	10,1 TPS	10,1 TPS

Table B.14: Fabric latencies for 10 TPS

Name	Max latency	Min latency	Avg latency
open	5,02 s	0,54 s	2,37 s
query	0,04 s	0,01 s	0,02 s

Table B.15: Fabric resource consumption for 10 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	236,9 MB	229,7 MB	111,72 %	32,02 %
query	236,0 MB	236,0 MB	7,59 %	6,34 %

Table B.16: Fabric data transfer for 10 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	1 270,3 KB	1 234,2 KB	180 KB	560 KB
query	372,1 KB	347,5 KB	0 KB	0 KB

B.2.2 20 TPS send rate results

Ethereum results at 20 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Ethereum/20 TPS/report.html`.

Table B.17: Ethereum summary results for 20 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	20,2 TPS	8,3 TPS
query	100	0	20,2 TPS	20,2 TPS
transfer	100	0	20,2 TPS	10,1 TPS

Table B.18: Ethereum latencies for 20 TPS

Name	Max latency	Min latency	Avg latency
open	7,20 s	3,02 s	5,05 s
query	0,07 s	0,01 s	0,02 s
transfer	7,07 s	3,02 s	5,04 s

Table B.19: Ethereum resource consumption for 20 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	716,5 MB	641,2 MB	96,19 %	22,86 %
query	756,1 MB	756,0 MB	89,54 %	42,34 %
transfer	757,0 MB	756,8 MB	62,15 %	30,05 %

Table B.20: Ethereum data transfer for 20 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	460,8 KB	289,7 KB	0 KB	0 KB
query	117,8 KB	42,5 KB	0 KB	0 KB
transfer	495,3 KB	254,9 KB	0 KB	0 KB

Fabric results at 20 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Fabric/20 TPS/report.html`.

Table B.21: Fabric summary results for 20 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	20,1 TPS	19,1 TPS
query	100	0	20,2 TPS	20,1 TPS
transfer	86	14	20,2 TPS	16,4 TPS

Table B.22: Fabric latencies for 20 TPS

Name	Max latency	Min latency	Avg latency
open	0,72 s	0,11 s	0,32 s
query	0,05 s	0,01 s	0,02 s
transfer	0,59 s	0,13 s	0,32 s

Table B.23: Fabric resource consumption for 20 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	235,5 MB	224,5 MB	60,78 %	23,42 %
query	244,8 MB	240,5 MB	32,11 %	23,03 %
transfer	250,8 MB	249,6 MB	68,43 %	52,50 %

Table B.24: Fabric data transfer for 20 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	2 027,2 KB	1 769,7 KB	580,0 KB	2 048,0 KB
query	526,6 KB	505,3 KB	0 KB	0 KB
transfer	1 896,1 KB	1 569,5 KB	196,0 KB	1 816,0 KB

Besu results at 20 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Besu/20 TPS/report.html`.

Table B.25: Fabric summary results for 20 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	20,3 TPS	10,3 TPS
query	100	0	20,2 TPS	20,1 TPS
transfer	99	1	20,2 TPS	16,5 TPS

Table B.26: Fabric latencies for 20 TPS

Name	Max latency	Min latency	Avg latency
open	6,05 s	2,10 s	4,18 s
query	0,14 s	0,02 s	0,02 s
transfer	6,16 s	1,04 s	3,37 s

Table B.27: Fabric resource consumption for 20 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	263,7 MB	256,6 MB	167,94 %	83,26 %
query	259,6 MB	258,3 MB	127,50 %	82,72 %
transfer	275,0 MB	267,5 MB	130,12 %	99,21 %

Table B.28: Fabric data transfer for 20 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	450,2 KB	306,1 KB	0 KB	0 KB
query	101,8 KB	34,4 KB	0 KB	0 KB
transfer	414,2 KB	176,8 KB	0 KB	0 KB

Iroha results at 20 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Iroha/20 TPS/report.html`.

Table B.29: Fabric summary results for 20 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	20,7 TPS	8,9 TPS
query	100	0	20,2 TPS	20,2 TPS

Table B.30: Fabric latencies for 20 TPS

Name	Max latency	Min latency	Avg latency
open	9,65 s	1,50 s	5,92 s
query	0,03 s	0,01 s	0,02 s

Table B.31: Fabric resource consumption for 20 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	240,5 MB	230,6 MB	46,33 %	14,88 %
query	235,5 MB	235,5 MB	12,57 %	7,78 %

Table B.32: Fabric data transfer for 20 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	1 279,3 KB	916,2 KB	328 KB	624 KB
query	261,9 KB	244,8 KB	0 KB	0 KB

B.2.3 50 TPS send rate results

Ethereum results at 50 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Ethereum/50 TPS/report.html`.

Table B.33: Ethereum summary results for 50 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	50,7 TPS	11,4 TPS
query	100	0	50,5 TPS	50,3 TPS
transfer	100	0	50,5 TPS	14,2 TPS

Table B.34: Ethereum latencies for 50 TPS

Name	Max latency	Min latency	Avg latency
open	7,11 s	2,05 s	5,40 s
query	0,10 s	0,01 s	0,02 s
transfer	6,22 s	5,03 s	5,58 s

Table B.35: Ethereum resource consumption for 50 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	720,6 MB	651,3 MB	90,01 %	29,57 %
query	750,7 MB	750,5 MB	47,13 %	37,65 %
transfer	751,5 MB	751,4 MB	67,79 %	32,79 %

Table B.36: Ethereum data transfer for 50 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	472,4 KB	263,1 KB	116,0 KB	0 KB
query	112,5 KB	40,6 KB	0 KB	0 KB
transfer	471,0 KB	182,6 KB	0 KB	0 KB

Fabric results at 50 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Fabric/50 TPS/report.html`.

Table B.37: Fabric summary results for 50 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	28,2 TPS	25,3 TPS
query	100	0	50,5 TPS	50,0 TPS
transfer	53	47	33,9 TPS	15,2 TPS

Table B.38: Fabric latencies for 50 TPS

Name	Max latency	Min latency	Avg latency
open	0,93 s	0,30 s	0,62 s
query	0,12 s	0,02 s	0,05 s
transfer	1,81 s	0,43 s	1,13 s

Table B.39: Fabric resource consumption for 50 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	281,2 MB	268,8 MB	73,77 %	17,16 %
query	322,1 MB	309,6 MB	23,75 %	23,75 %
transfer	320,8 MB	320,2 MB	66,66 %	40,15 %

Table B.40: Fabric data transfer for 50 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	1 288,3 KB	1 144,5 KB	84,0 KB	1 012,0 KB
query	0 KB	0 KB	0 KB	0 KB
transfer	1 481,8 KB	1 130,1 KB	4,0 KB	1 048,0 KB

Besu results at 50 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Besu/50 TPS/report.html`.

Table B.41: Fabric summary results for 50 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	41,9 TPS	10,3 TPS
query	100	0	50,6 TPS	50,2 TPS
transfer	100	0	49,6 TPS	14,3 TPS

Table B.42: Fabric latencies for 50 TPS

Name	Max latency	Min latency	Avg latency
open	8,61 s	3,98 s	6,07 s
query	0,19 s	0,02 s	0,05 s
transfer	6,00 s	1,47 s	4,60 s

Table B.43: Fabric resource consumption for 50 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	259,9 MB	221,0 MB	146,25 %	70,75 %
query	251,4 MB	250,2 MB	155,77 %	78,58 %
transfer	269,8 MB	256,9 MB	135,93 %	82,82 %

Table B.44: Fabric data transfer for 50 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	434,8 KB	279,5 KB	0 KB	0 KB
query	133,6 KB	48,4 KB	0 KB	0 KB
transfer	461,2 KB	214,8 KB	0 KB	0 KB

Iroha results at 50 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Iroha/50 TPS/report.html`.

Table B.45: Fabric summary results for 50 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	42,9 TPS	8,8 TPS
query	100	0	50,5 TPS	50,1 TPS

Table B.46: Fabric latencies for 50 TPS

Name	Max latency	Min latency	Avg latency
open	11,38 s	6,75 s	9,51 s
query	0,05 s	0,01 s	0,02 s

Table B.47: Fabric resource consumption for 50 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	244,9 MB	235,5 MB	59,48 %	9,36 %
query	243,2 MB	243,0 MB	39,87 %	14,73 %

Table B.48: Fabric data transfer for 50 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	559,5 KB	557,5 KB	420 KB	168 KB
query	345,3 KB	326,9 KB	60 KB	0 KB

B.2.4 100 TPS send rate results

Ethereum results at 100 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Ethereum/100 TPS/report.html`.

Table B.49: Ethereum summary results for 100 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	102,8 TPS	13,1 TPS
query	100	0	100,7 TPS	93,7 TPS
transfer	100	0	73,5 TPS	16,2 TPS

Table B.50: Ethereum latencies for 100 TPS

Name	Max latency	Min latency	Avg latency
open	7,47 s	5,73 s	6,48 s
query	0,17 s	0,01 s	0,06 s
transfer	6,17 s	4,11 s	5,07 s

Table B.51: Ethereum resource consumption for 100 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	950,5 MB	808,0 MB	99,87 %	36,60 %
query	951,0 MB	950,8 MB	10,19 %	5,20 %
transfer	992,1 MB	988,4 MB	83,62 %	43,75 %

Table B.52: Ethereum data transfer for 100 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	437,6 KB	207,3 KB	228,0 KB	0 KB
query	17,5 KB	4,8 KB	64,0 KB	0 KB
transfer	467,1 KB	241,9 KB	372,0 KB	0 KB

Fabric results at 100 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Fabric/1000 TPS/report.html`.

Table B.53: Fabric summary results for 100 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	28,4 TPS	25,5 TPS
query	100	0	66,7 TPS	65,2 TPS
transfer	51	49	31,2 TPS	13,2 TPS

Table B.54: Fabric latencies for 100 TPS

Name	Max latency	Min latency	Avg latency
open	0,86 s	0,33 s	0,56 s
query	0,22 s	0,03 s	0,10 s
transfer	1,67 s	0,41 s	1,12 s

Table B.55: Fabric resource consumption for 100 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	190,1 MB	181,0 MB	73,38 %	21,74 %
query	206,5 MB	203,1 MB	65,83 %	35,46 %
transfer	209,9 MB	208,3 MB	82,62 %	50,56 %

Table B.56: Fabric data transfer for 100 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	1 616,3 KB	1 404,8 KB	188,0 KB	1 508,0 KB
query	440,4 KB	448,9 KB	0 KB	0 KB
transfer	1 853,9 KB	1 495,8 KB	88,0 KB	1 576,0 KB

Besu results at 100 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Besu/100 TPS/report.html`.

Table B.57: Fabric summary results for 100 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	102,8 TPS	10,0 TPS
query	100	0	101,2 TPS	68,5 TPS
transfer	99	1	86,4 TPS	12,1 TPS

Table B.58: Fabric latencies for 100 TPS

Name	Max latency	Min latency	Avg latency
open	9,23 s	4,98 s	6,58 s
query	0,82 s	0,25 s	0,56 s
transfer	7,51 s	2,50 s	4,32 s

Table B.59: Fabric resource consumption for 100 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	269,7 MB	266,0 MB	161,79 %	77,34 %
query	257,8 MB	257,8 MB	0,36 %	0,36 %
transfer	287,4 MB	286,2 MB	109,90 %	66,13 %

Table B.60: Fabric data transfer for 100 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	392,9 KB	260,9 KB	0 KB	0 KB
query	0 KB	0 KB	0 KB	0 KB
transfer	441,4 KB	282,4 KB	0 KB	0 KB

Iroha results at 100 TPS send rate

The results file for the following data is to be found in the *electronic attachment* at `thesis/results/Iroha/100 TPS/report.html`.

Table B.61: Fabric summary results for 100 TPS

Name	Succ	Fail	Send Rate	Throughput
open	100	0	42,5 TPS	9,1 TPS
query	100	0	84,5 TPS	83,5 TPS

Table B.62: Fabric latencies for 100 TPS

Name	Max latency	Min latency	Avg latency
open	10,96 s	6,43 s	9,06 s
query	0,09 s	0,01 s	0,04 s

Table B.63: Fabric resource consumption for 100 TPS

Name	RAM (max)	RAM (avg)	CPU (max)	CPU (avg)
open	241,1 MB	230,3 MB	152,24 %	32,43 %
query	238,1 MB	238,1 MB	0,23 %	0,23 %

Table B.64: Fabric data transfer for 100 TPS

Name	Traffic In	Traffic Out	Disc Read	Disc Write
open	1115,2 KB	1 086,7 KB	0 KB	552 KB
query	0 KB	0 KB	60 KB	0 KB