Implementation of a Blockchain-Based V2V Energy Sharing Application using Byzantine Fault Tolerant Consensus Algorithms

Sadhil Jindal

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

In this dissertation, Byzantine Fault Tolerance (BFT) consensus methods are used to construct a Blockchain-based V2V energy sharing application. This study's main goal is to assess the efficiency and performance of the SBFT and Linear BFT consensus types within the framework of the V2V energy sharing application. A proof-of-concept Blockchain-based V2V energy sharing system was designed and developed for the project, and a number of experiments were run to assess its performance in terms of latency, throughput, and scalability. The study's findings show that, in comparison to regular BFT, the SBFT and Linear BFT consensus types can considerably increase the effectiveness and dependability of the V2V energy sharing application, as well as achieve higher transaction throughput and reduced latency. Overall, this study sheds light on the potential of BFT consensus algorithms and Blockchain technology to enable safe and effective energy sharing in a V2V environment.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Sadhil Jindal*)

Acknowledgements

I would like to thank my supervisor, Professor Xiao Chen, for his support throughout this project. Additionaly, I would like to thank Dr Mary Cryan for her advice during our meeting. I would also like to thank Meng Kun for his support durling the implementation phases of this project.

Thank you to my parents Sunita and Sushil Jindal for their love and support during my time at university.

Table of Contents

Chapter 1

Introduction

There are now more electric vehicles (EVs) on the road as a result of the global transition towards renewable energy technology. These EVs have the capacity to both use and distribute energy. As a result, new possibilities for peer-to-peer energy trading have emerged, allowing owners of EVs to sell their extra energy to other users who have higher need. However, peer-to-peer energy trading using electric vehicles is not supported by the current energy infrastructure, which has prompted the development of new technologies like Blockchain.

Blockchain [10] technology is a good contender for energy trading applications because it offers a decentralised platform for safe and open peer-to-peer transactions, which eliminates the need for a governing authority that has total control over the system increasing fairness and transparency. In particular, Byzantine Fault Tolerance (BFT) [18] consensus algorithms can guarantee the security and dependability of the system while facilitating quick and effective transactions in Blockchain-based energy trading systems.

The challenges of implementing and using a Blockchain-based application is that a large number of nodes in the network bottleneck the consensus system to a point where using a centralised application would be a much better choice, even with its weaker security. In recent times however there has been a lot work and research on consensus algorithms and faster algorithms have been created such as Linear BFT [27] and SBFT [22].

The goal of this project is to design and implement a Blockchain-based V2V energy trading framework that uses various BFT consensus variations BFT, LinearBFT and SBFT. The proposed system enables EV owners to trade energy with nearby consumers. The system uses a BFT consensus technique to assure the transactions' security and dependability while also enabling quick and effective transaction processing. The main contribution of this project is the utilisation and testing of the improved BFT consensus variations Linear BFT and SBFT. The application was made modular from the consensus to allow easy rotation between the three different BFT consensus types.

The remainder of this report is structured as follows. In the next chapter, the related work in Blockchain-based energy trading is discussed, followed by the background

knowledge needed to understand this project. Following this, the the proposed V2V energy trading system is described and analysed in detail, including the system architecture, request processing and implementation. Next, the results of simulation experiments that demonstrate the effectiveness of the system are presented. Finally, the findings are laid out and opportunities for future work are discussed before concluding the project.

Chapter 2

Related Work

The field of Blockchain-based V2V [15] energy trading is an emerging area of research and development, with a growing amount of literature and real world solutions exploring the potential of this technology to transform the energy sector. In this section, some of the existing literature and solutions in this area are analysed.

2.1 Live Solutions for V2V Energy Trading

At present, there are only some existing projects regarding blockchain based EV energy sharing that are in the early stages of development, given that this is a new market.

One example of a V2V energy trading solution is The German firm eCharge Network [30], which seeks to establish a peer-to-peer network for electric vehicle (EV) charging. With the help of the platform, EV owners can lend out their charging stations to other EV owners and earn credits that can be used to the cost of charging at other stations around the network. Blockchain technology is used by the eCharge Network to speed up transactions and maintain the system's security and transparency. eCharge Network uses an Ethereum [6] Blockchain, compared to the scope of this project where the use of BFT consensus is explored.

Another instance is the European Union-funded CityFlow [20] project, which intends to develop a V2V energy trading market for electric vehicles. The platform would make it possible for EV owners to exchange energy with one another and the grid, balancing supply and demand and lowering energy costs. Although the CityFlow project is currently in the testing stage, successful trials have been carried out in a number of European towns. However, this energy trading project does not use Blockchain technology.

2.2 Prototype Projects and Literature

The Consensus Mechanism for Blockchain-Enabled Vehicle-to-Vehicle Energy Trading in the Internet of Electric Vehicles [28] is one of the frameworks that has been published. It suggests using a Byzantine Fault Tolerance (BFT) consensus method to make energy

trading in the Internet of Electric Vehicles (IoEV) [19] easier. Even in the face of malevolent actors, the BFT consensus algorithm ensures that all nodes in the network concur on the legitimacy of transactions, maintaining the system's integrity and security. However this framework does not investigate newer BFT consensus variations, instead it only uses the outdated Practical Byzantine Fault Tolerance (PBFT) [9] variant.

The Fast and Secured Vehicle-to-Vehicle Energy Trading Based on Blockchain Consensus in the Internet of Electric Vehicles [31] also makes a proposal for a fast and secure V2V energy trading system that makes use of a hybrid consensus algorithm combining Proof-of-Work (PoW) [21] and PBFT consensus protocols. Fast transaction validation is guaranteed by the hybrid consensus algorithm, which also keeps the system's decentralisation and security intact. However, the hybrid model of PoW and PBFT is still energy inefficient [11] compared to a pure BFT consensus variation, which is what this project focuses on.

The Efficient Vehicle-to-Vehicle (V2V) Energy Sharing Framework [29] is a different suggested framework that makes use of smart contracts to enable peer-to-peer energy trading between EVs. With transactions being verified and recorded on the Blockchain, the framework enables EV owners to share excess energy with other EVs in the network. This paper does not investigate BFT style consensus however, which is an area of research this project focuses on.

2.3 Summary

Overall, these works demonstrate the potential of Blockchain-based V2V energy trading systems, and highlight the advantages of using Blockchains, in that they ensure security [32] and reliability [26]. However, there is still much work to be done in this area, particularly in terms of scalability and real-world implementation.

Chapter 3

Background Research

In this chapter, the background for this project is discussed. This starts with an overview of Blockchains and a brief explanation of its key concepts. This is followed by a discussion on Permissioned and Permissionless Blockchains and an analysis on the uses and advantages for each. Byzantine Fault Tolerance is discussed as it is an important feature of modern Blockchains and distributed systems. This will lead to an analysis of BFT consensus algorithms. The next section will explain the Smart Byzantine Fault Tolerance library in detail.

3.1 Blockchain

3.1.1 Overview

Blockchain [21] technology has provided a means to create a distributed consensus about the state of a system. The idea of a 'Blockchain' was first proposed in 1979 by a doctoral candidate David Chaum in his research paper [10]. This work was not focused towards digital currencies, but instead just a general idea of how distributed databases should work and how one would go about creating one.

It was not until 2009 that the first modern, fully functioning secure Blockchain system was implemented. This was the infamous Bitcoin Blockchain that was based on Satoshi Nakamoto's work on peer-to-peer payment technologies. This system was a distributed peer-to-peer payment system that guaranteed anonymity and security (barring certain attacks) by creating a consensus on the order of transactions. The currency of this system was given after its own name Bitcoin [21]. Since its introduction there has been huge amounts of work and research in the field of Blockchains and Distributed Systems [17], as their usefulness has become more widely recognise. This work has resulted in a vast amount of new Blockchains being developed for various unique purposes apart from simple peer-to-peer payments.

A Blockchain is essentially an append only data ledger [25] set that is split into a chain of multiple data blocks. Blocks can be created and added to the end of the chain, but they can never be modified or removed. Each block i contains multiple data entries (usually transactions), the hash of the previous block $i-1$, and a nonce n that is utilised

to verify the hash of the previous block. Having each block constrain the hash of the previous block ensures that the Blockchain data cannot be modified. This is because, if the data contained within block i was changed, this would affect the hash of block i, leading to inequality between the hash value contained in block $i+1$. This would affect all blocks after block $i+1$ as the hash value corresponding to the previous block would change for each block as it is directly dependant on the data contained within the previous block. The entire chain of blocks comprises of the full data ledger.

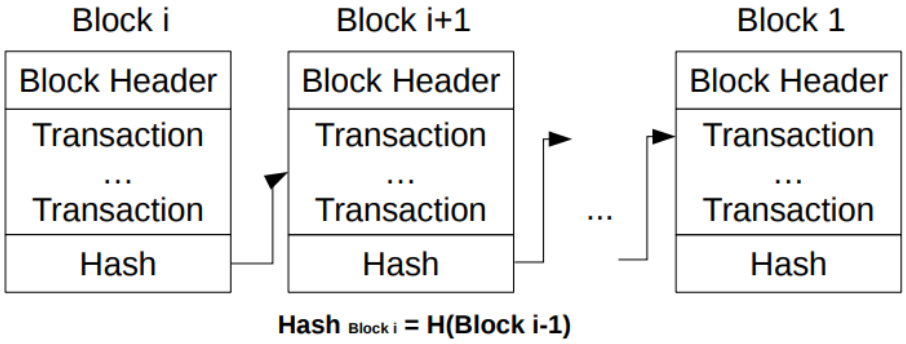

Figure 3.1: Example Blockchain

In modern times, distributed Blockchain systems have become increasingly popular for various uses such as peer to peer payments and distributed applications. In these distributed systems, nodes are responsible for maintaining the functionality of the network, and validating and adding data to the Blockchain. The validation of the data is carried out through a consensus [13] between all nodes on the system, to see if there is a majority agreement on the state of the Blockchain. Proof of Work and Proof of Stake [6] are popular consensus protocols while Byzantine Fault Tolerance is the consensus that is central to this project.

3.1.2 Permissioned and Permissionless Blockchains

Blockchain systems can be split into two categories, Permissioned [8] systems and Permissionless systems.

In Permissionless Blockchain systems, also known as public Blockchain systems, there is no restriction on who can or cannot join the network to use the core functionality or participate in the consensus to append data to the system ledger. In Permissionless systems, there needs to be a form of incentive [7] for nodes to join the network and participate in the consensus otherwise there will not be enough nodes to achieve a safe consensus. For example, if there are only n nodes participating in the consensus of a network where n is relatively low, then an attacker could push $n+1$ malicious nodes into the network to gain full control of the consensus as it only need 51% of the voting power to manipulate the result in its favour (also known as the 51% attack [16]). Thus, nodes are rewarded for participating in the consensus of the network and growing the network's Blockchain. Proof of Work (PoW) and Proof of Stake (PoS) are two renowned consensus algorithms that reward nodes for participating in the consensus of

the network. An interesting point is PoW is notorious for its energy inefficiency, and while PoS is more energy efficient, it is not as secure as PoW. Since Permissionless Blockchains need to reward nodes for their participation, each consensus algorithm has its own drawbacks in order to ensure that nodes are rewarded for their participation.

In Permissioned Blockchain systems, there is a system owner or operator is responsible for deciding who can and cannot join the network to use the core functionality or participate in the consensus to append data to the system ledger. The owner or operator can modify the Blockchain as they see fit. This type of Blockchain system is a suitable solution for businesses looking to automate certain processes such as managing supply chains, creating contracts and verifying payments between parties. Popular Permissioned Blockchain examples are R3 Corda [24] and HyperLedger Fabric [8]. Since this Blockchain is not open for anyone to join, the nodes on the system network are usually ones that have been deployed by the owner or operator or a third party that is taking part in the business activities specified by the functionality of the Blockchain. Nodes now do not need an incentive to join the network, as they are now deployed by businesses or other entities for the specific purpose of keeping the functionality of the Blockchain available. This gives the option of using other consensus algorithms that do not have mechanisms for rewarding nodes such as Byzantine Fault Tolerance [9] which is the consensus algorithm that is central to this paper.

3.2 State Machine Replication

State machine replication (SMR) [12] is a distributed computing technology that offers fault tolerance and scalability by enabling several replicas of a system to operate cooperatively. It makes sure that each replica maintains the same state and that each replica processes requests in the same sequence. Treating the service like a deterministic [14] state machine is the fundamental tenet of SMR. A mathematical model known as a state machine consists of a limited number of states and a set of possible transitions between them. Each transition results in an output event, a new state, and is initiated by an input event. If a state machine generates a distinct output event and a new state from a given input event and existing state, it is deterministic.

Each replica in SMR keeps a separate copy of the state machine. Replicas receive client requests and process them in the same sequence. SMR makes use of a consensus technique to guarantee that all replicas process requests in the same sequence. A client's request is initially sent to the primary replica when sending one to the service. Requests are ordered by the primary replica and sent to the other replicas in that same order. Following the same order of request execution, the copies provide the identical output events and state changes. The primary replica notifies the client when all replicas have completed processing a request.

SMR offers a number of advantages. First, by replicating the service over several nodes, it offers fault tolerance. The other nodes can carry on providing the service even if one fails. It also offers scalability by enabling many nodes to handle queries concurrently. By making sure that all copies keep the same state and handle requests in the same sequence, it also ensures consistency.

One of the issues with SMR is making sure the state machine is still deterministic in the presence of faults. For example, if a node crashes and subsequently restarts, it may have missed some requests and state changes. To ensure that the node can catch up, it must be able to recover its state by sending the requests it missed. This requires that the state machine be deterministic and that the requests be idempotent, or that they return the same result when done repeatedly. Another drawback of SMR is the need to maintain the effectiveness and scalability of the consensus procedure. Consensus protocols are frequently designed to work with a small number of nodes; as the number of nodes increases, the performance of the protocols may degrade.

3.3 Byzantine Fault Tolerance

3.3.1 Introduction

Byzantine Fault Tolerance is the property of a distributed system that allows it to continue operating correctly even when certain nodes in the system are acting incorrectly or maliciously. This property is derived from the Byzantine Generals' Problem [18]. This problem is a model of generals each with their own army that are trying to decide whether to attack or retreat from the battle. The generals can only communicate with each other via a messenger. There is no guarantee that the messengers will reach the other general, and generals can act maliciously and send incorrect messages, see figure reffig:generals. It has been proven previously and well known that if there are f byzantine generals (malicious generals or using unreliable messengers) then there must be greater than $3f + 1$ generals in total to reach the correct decision in the consensus.

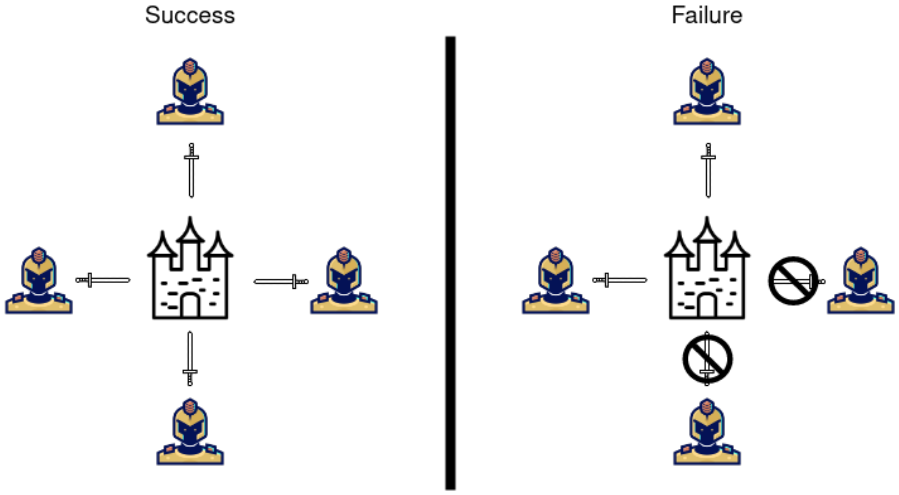

Figure 3.2: BFT Generals' Problem

This problem can be applied to a distributed computer system. The generals are represented as nodes and messengers are represented by the underlying point to point communication service. All nodes on the system are required to reach a consensus on a particular topic, however nodes can crash or act maliciously so this adds a layer of complexity to the system. Byzantine Fault Tolerant is the category of systems that are

not susceptible to this problem and are able to reach consensus even with f malicious or faulty nodes, meaning they employ some kind of Byzantine Fault Tolerant Algorithm.

There are many different Byzantine Fault Tolerant consensus algorithms such as the renowned PoW and PoS that are utilised by the two largest cryptocurrency networks, Bitcoin and Ethereum respectively. The algorithms that will be focused on in this project are BFT, Linear BFT and SBFT.

3.3.2 Practical Byzantine Fault Tolerance

PBFT [9] very first in its class of consensus algorithms when it was introduced in 1999. It was the first State Machine Replication algorithm to allow a set of server replicas to come to consensus on the state of the system after executing client instructions. It guarantees both safety and liveliness under the condition that no more than $(n-1)/3$ replicas are faulty when there are n replicas in the system. Safety implies that the system satisfies the linearizability property, meaning externally it should seem as if the distributed system is a monolithic system that will only converge to one outcome. Liveliness is defined as the property that clients will eventually receive a response to their requests given that no more than $(n-1)/3$ replicas are faulty and $delay(t)$ does not grow faster than t indefinitely, where $delay(t)$ is the time between sending a message and the message reaching its destination. Most system networks have some sort of guarantee on the delivery of messages, for example the TCP Network Protocol [23] ensures that data is received at the destination given a reliable underlying IP channel.

The PBFT algorithm is based on the architecture of a client and a network of server replicas, where the client sends a request into the network. The requests are often some form of transactions between two or more parties. The request is validated through the network nodes exchanging messages to agree on the order of execution of client requests. If the order of execution of client requests is the same for all nodes, it is guaranteed that the state of the system will be synchronized as all nodes execute their requests in the exact same order. After a consensus has been reached within the network, the result of the client request is sent back to the client. Each consensus instance is performed under a **view**, the state of the network at that moment in time which consists of the leader node and replicas. Views can change if the leader node fails or is unresponsive.

Pre-Prepare. The client sends the request into the network, and the request is picked up by the current primary node. The primary node broadcasts a *Pre-Prepare* message $<< pre\_prepare, v, n, d >_{\sigma i} m >$ where m is the client's message, v is the view number, n is the sequence number, d is the message digest $hash(m)$ and σi denotes the signing of the message with the private key of replica i. Other replicas receive the message, verify the legitimacy of d, the signature of the message is correct and the replica itself is in view v. If these checks are passed, replicas add this message to their logs and then broadcast a *Prepare* message $< prepare, v, n, d >_{\sigma i}$ to ALL replicas.

Prepare. When a replica receives a $< prepare, v, n, d >_{\sigma i}$ message it verifies the signature is correct and that v matches the replica's current view. It adds each valid prepared message to its log. Once a replica has received $2f+1$ (each replica sends to itself too) valid *Prepare* messages, the replica broadcasts the *Commit* message

$< commit, v, n, D(m), i >_{\sigma i}$ where i is the identity of the replica that is broadcasting the message and D(m) is the message applied to the digest function.

Commit. When a replica receives $< commit, v, n, D(m), i >_{\sigma i}$ message it again ensures that it the signature is valid and it is in the same view and following these checks the message is added to the log. When the replica has received $2f + 1$ valid *Commit* messages, it can then execute the operation in the message associated with the corresponding view and sequence number. If all replicas agree on the order of execution of client requests, it is trivial that all replicas will have a synchronised state as execution of the same requests in the same order will result in identical states and outputs.

The *Pre-Prepare* and *Prepare* phases ensure that non-Byzantine replicas agree for the requests within a view. The commit phase along with the view change protocol ensure non-Byzantine replicas all have the same sequence numbers for client requests even if the committed local state is achieved in different views.

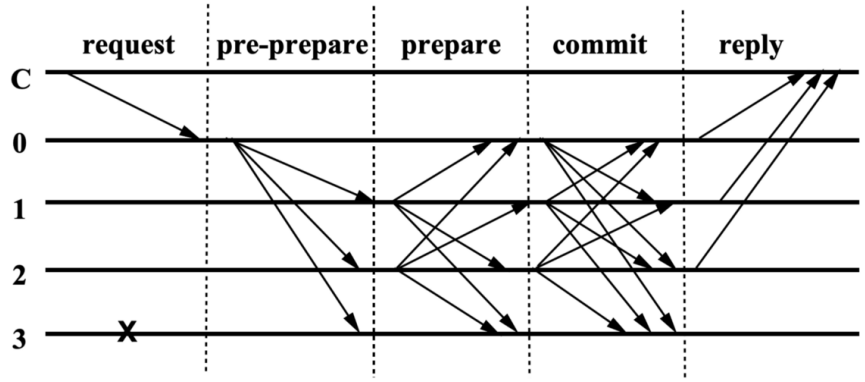

Figure 3.3: PBFT Execution Cycle

In figure 3.3 it can be seen that replica 3 becomes byzantine, but the execution of PBFT still continues correctly as there are $3f + 1 = 4$ nodes. Notice the all-to-all communication in the *Prepare* and *Commit* steps. This communication is of order $O(n^2)$ where n is the number of replicas, which is expensive and in some cases can bottleneck the underlying communication system.

3.3.3 Linear BFT

The Linear BFT [27] consensus algorithm is a variation of BFT. The key to this algorithm is the use of a Threshold Signature Scheme (TSS) [3]. TSS is a method to generate a signature that is actually comprised of several signatures. A threshold can be set on the amount of signatures that need to be present in the combined key for it to pass verification. In mathematical notation, an (n,t)-threshold signature on a message m is a signature that only succeeds in verification if at least $t + 1$ participants sign m.

As highlighted previously, in the *Prepare* and *Commit* steps of PBFT there is $O(n^2)$ all-to-all communication. Linear BFT can use a TSS to reduce the communication complexity from $O(n^2)$ to $O(n)$. The primary node acts as a collector for the *Prepare* and *Commit* phases. Instead of each replica broadcasting their messages to every other

replica, replicas only send their messages to the primary. The collector can then create a $(n, 2f+1)$-threshold signature ts from all the individual signatures that were received from other replicas. The primary sends this signature back to all replicas, who already possess the public key. This signature will only pass verification if there were at least $(n, 2f+1)$ replicas who sent a *Prepare* or *Commit* message, since that is what the threshold was set to.

This modification adds two additional steps, by splitting *Prepare* and *Commit* into two separate steps. After a replica receives a *Pre-Prepare* message, it only sends its *Prepare* message to the collector. The collector can then generate the threshold signature from all *Prepare* messages, and broadcast it to all replicas. Similarly, replicas send their *Commit* messages to the collector again. The collector once again generates a threshold signature from the *Commit* messages and broadcasts the signature to all replicas. Replicas can then send their reply to the client as in the normal case.

This version of BFT has additional communication steps, but makes the communication much more efficient. This is because, when the network size grows, having a quadratic complexity like for message exchange like PBFT becomes increasingly inefficient. Linear BFT allows for larger network size without sacrificing the latency and throughput of consensus taking place over the network. However, there are drawbacks to using TSS too, as the setup costs for this are quite expensive. TSS necessitates unique public and private key pairs that are correlated. However, generating these key pairs in a decentralized environment requires a distributed key generation (DKG) [4] protocol, which involves significant communication. One example of a DKG is the Joint-Feldman algorithm, which has communication footprint of $O(n^3)$. The most commonly used DKG for Linear BFT is described in reference, and has a $O(nlog(n)^k)$ communication footprint. If the set of nodes in the system changes, then a new set of public and private key pairs must be generated to ensure that the threshold signature scheme remains secure. This typically involves running the DKG protocol again with the new set of nodes, which can be a time-consuming and communication-intensive process.

3.3.4 Scalable Byzantine Fault Tolerance

Scalable Byzantine Fault Tolerance (SBFT) [22] is a further extension on BFT from Linear BFT. This SBFT takes advantage of TSS the same way that Linear BFT does. To extract more performance from the network, this consensus algorithm exploits the situation where all nodes in a network are stable, meaning there are $3f+1$ stable nodes. When the whole network is stable, and each replica on the ordering of a client request in the *Pre-Prepare* phase, it is not necessary for a *Prepare* step. In SBFT, when the nodes are stable and agree on the order of the client request in the *Pre-Prepare* step the *Prepare* step can be skipped, and the consensus can proceed to the *Commit* phase. This reduces the amount of messages exchanged and will increase the performance of the consensus by a considerable amount. This is referred to as the *Fast Path* in SBFT.

In the case where nodes in the network become unstable, meaning there are $1 \leq x \leq f$ x Byzantine nodes in the system, the SBFT algorithm can revert to the *Slow Path*. This is essentially just Linear BFT. The switch to the *Slow Path* is triggered when primary receives less than $3f+1$ *Sign Share* messages and the timer to receive all

$3f + 1$ messages has expired. This timeout value can be adjusted according to network conditions. The primary node can then assemble and broadcast the *Prepare* message as described in Linear BFT. The following phases are the exact same as Linear BFT, and this is what the *Slow Path* consists of. This switch between the two paths allows the consensus to make progress faster when all nodes in the network are stable, and more importantly it allows for progress even when there are up to f Byzantine nodes which is our basis for all BFT algorithms.

Given certain scenarios, the performance of SBFT can be improved and it can made more resilient by adding redundant nodes to the network. If in a certain consensus network, there is capacity and resources to add more nodes to the network, it allows for more padding of when the *Fast Path* can be used. Adding $2c$ redundant nodes to the system creates the property that the *Fast Path* can be used even when their are $c \leq f$ Byzantine nodes.

SBFT suffers from the same drawbacks as Linear BFT with the cost of setting up a new TSS every time the network graph changes. The use case for SBFT is targeted at networks where it is known that the nodes are usually stable so the *Fast Path* can be used more often to progress consensus instances more efficiently.

3.4 BFT_SMaRt

BFT_SMaRt [2] is an implementation of PBFT in Java that will be used mainly in this project. This code base provides a basic consensus framework based on PBFT with some modern features such as multicore awareness, code modularity and node reconfiguration support. This implementation has been proven to achieve significantly better results than previous outdated implementations of PBFT. This baseline standard BFT_SMaRt implementation will be referred to as BFT.

The most significant advantage of BFT_SMaRt is that the consensus functionality is separated from the SMR functionality. This allows for decentralized applications to be easily constructed on top of BFT_SMaRt by modifying the SMR part of the code base. The basic consensus can be adapted to recreate other BFT consensus types such as Linear BFT [27] and SBFT [22]. This code base also has modular components for the underlying point to point communication service, reconfiguration of nodes in the network, and state transfer to update the state of new or faulty nodes in the network to match the true state of the system. See figure

Due to BFT_SMaRt's focus on modularity, the code base can be easily modified to recreate other versions BFT. There exists an implementation of Linear BFT that exploits the modularity of BFT_SMaRt by modifying the code base to use TSS. This is the implementation that will be used in this project for running application on a Linear BFT consensus. Similarly, an existing implementation of SBFT running on BFT_SMaRt was available to use. As mentioned, using BFT_SMaRt as the code base for all three consensus variations to be used in this project allows for the exact same application to run on top of each consensus system, without needing to modify the code base.

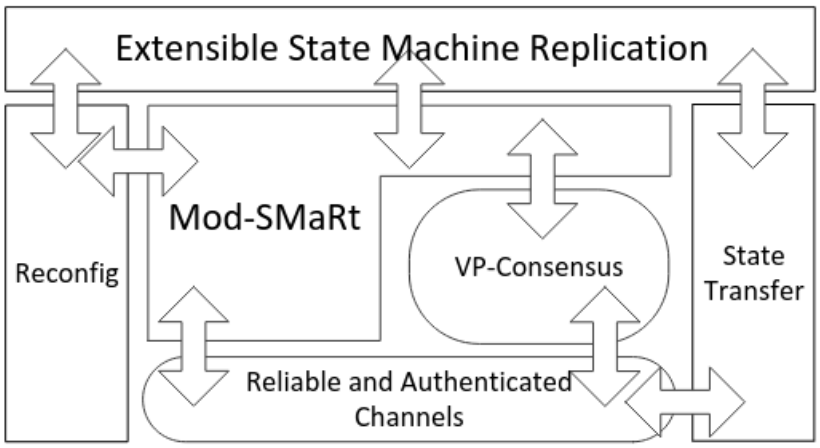

Figure 3.4: BFT_SMaRt Model

3.5 Vehicle to Vehicle Charging

Electric vehicles (EVs) can share electricity with each other directly through V2V charging [15], which eliminates the need for a charging station or the power grid. In V2V charging, one EV serves as the donor car and supplies electricity from its battery to another EV, referred to as the recipient vehicle. A physical connection, such as a cable, or wireless power transfer technology, such inductive or magnetic resonance coupling, can be used to accomplish this.

V2V charging provides a number of potential advantages, such as allowing EVs to increase their driving range by ”refuelling” at other EVs or to share power in emergency circumstances where grid power may be absent or unstable. Additionally, it might lessen the demand for charging infrastructure and give EV owners additional revenue streams, such selling surplus energy to the grid or to other EV drivers. The standardisation of connection protocols and assuring the security and dependability of power transfer are two issues that V2V charging must address. Nevertheless, a number of businesses and research institutions are working hard to develop and test V2V charging systems, and they may play a significant role in the development of the EV ecosystem in the future.

Chapter 4

System Model

4.1 Scenario

Electric vehicles (EVs) are becoming popular due to several factors, including their reduced environmental impact, cost savings in the long run, advancements in technology, and government policies. With the explosive growth in this market that is yet to come, there are problems that need to be solved to withstand the resulting energy demand. Two of the main concerns that are addressed by this project are:

- The distribution of EV charging points in certain areas can be sparse. Even in some densely populated areas around the world, there is not enough EV charging infrastructure to ensure that their is charging solutions available to support a large amount of EV usage.
- EV owners may be left stranded when they do not have sufficient charge left in their battery to complete the remaining journey to their destination or even the nearest EV charging point.

These shortcomings create a demand for EV owners to share energy amongst themselves when charging stations are not available due to one of the mentioned reasons. This demand is the perfect use case for V2V charging , which is a mobile charging solution where EVs can transfer energy wirelessly to other EVs. The network of EV charging points is slowly expanding to meet the predicted demand when EVs become mandatory, but V2V will still be a key technology in the situation that stationary charging points are not accessible and a more mobile solution is required. This project prototypes an application that could be used for EV owners to engage in V2V transactions.

The application scenario comprises of EV owners at their respective current locations and meeting points. The meeting points are at fixed locations that act as synthetic charging points. Energy consumers and providers will meet at the optimal meeting point when a suitable transaction has been submitted. EV owners who have a shortage of battery charge can send requests via the application, similarly EV owners who have a surplus of energy and are willing to sell this energy can submit requests via the application. The system then computes what consumer, provider, and meeting point combinations yield the lowest cost for the consumer and highest profit for the provider

based on variables that are discussed in detail later. The system then instructs and gives directions to the designated meeting point to both consumer and provider. At arriving at the specified meeting point, the provider should use V2V charging to transfer the previously agreed amount of energy.

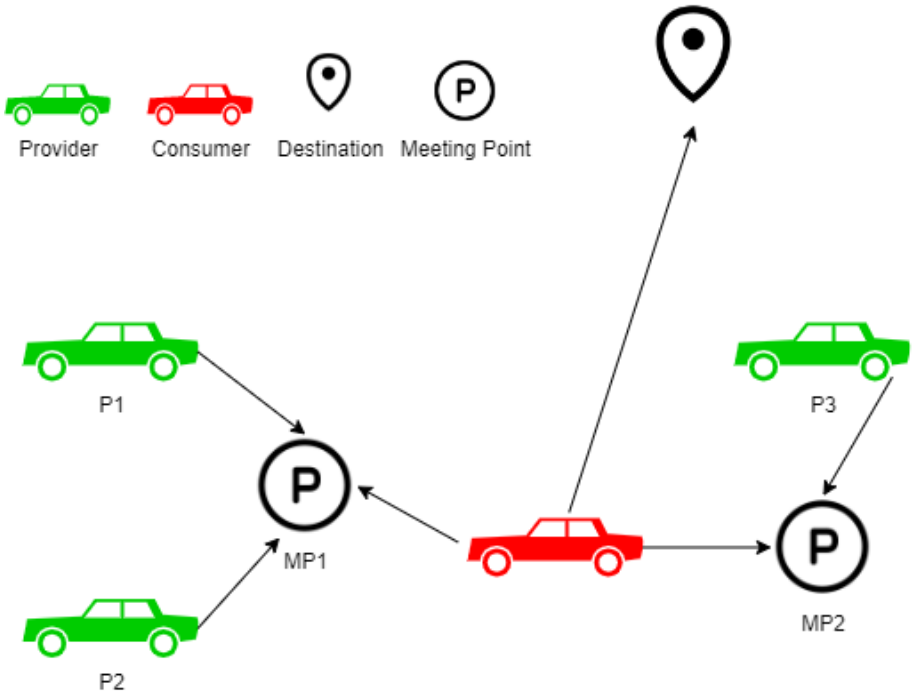

Figure 4.1: Application Use Scenario

A concrete user scenario can be explained by using figure 4.1. Three EV owners have a surplus of energy and are going to be within the same area for some amount of time. They take the opportunity to make some extra money by using the application to sell their surplus energy for profit. The three providers submit their respective requests to the system and the state is updated accordingly to reflect the prices each provider is willing to trade at. After some time, an EV owner nearby realises that they do not have sufficient charge to be able to complete their journey to their destination and their EV does not have sufficient charge to reach the nearest charging point. The stranded owner realises that it can make use of this application to attempt to purchase energy. The consumer submits a request to the system for exactly how much charge they require to reach their destination safely. The state of the system is updated accordingly to reflect this new request. The underlying system logic computes which provider, if any, leads to the minimum cost for the consumer, and also the maximum profit for the provider. In this scenario, provider 3 is decided to be the best match for the consumer. The transaction is executed and the relevant information should be updated in the system state, such as the transferring of currency from consumer to provider. Both the consumer and the provider drive to the designated meeting point (meeting point 2). Upon arriving at the destination, the provider uses their EV to supply the consumer with the agreed upon amount of energy, using V2V charging technology. The EVs of the provider and consumer will send the system updates on their battery state once the charging has been completed to verify that the exchange has been successful. The consumer now has

enough charge in their EV to travel to their destination, and the provider has made some profit from the transaction, meaning both parties have received some type of benefit from the application.

This process describes the use case for this system from the point of view of clients of the system. This scenario needs to be investigated in detail to extract all variables that need to be taken into account while matching providers and consumers. In the next section the requirements of the system are extracted from this use case.

4.2 Requirements

This application will be a Consumer-to-Consumer (C2C) service that allows clients to participate in an energy exchange with other clients to buy and sell energy for their EVs. As this project is focused on consensus and Blockchain technology, the requirements of this application will be focused on these aspects rather than considering the entire system end-to-end. As a result, requirements such as User Experience/Interface, Underlying Communication Channels, Physical Infrastructure and anything not related to the core of this project will mot be considered in the requirements or design of the application.

The requirements set to be achieved in this application are as follows:

- *Basic Request Functionality* - Clients should be able to send Buy/Sell requests into the system with all required information. Notifications should be delivered to clients upon successful requests.
- *Deterministic Buy/Sell Request Matching Algorithm* - Client Buy and Sell requests should be matched via the implemented algorithm that ensures satisfaction for both providers and consumers. Providers should not be selling for a loss and consumers should be provided with the best deal on the market, considering their location and other factors discussed later. Matches should be deterministic to allow for State Machine Replication.
- *Request Validation and Ordering* - All client requests should be validated and ordered via consensus before they are confirmed and added to the system. This process should be secure and transparent so that malicious parties cannot tamper with requests and requests can be verified by interested third parties.
- *State Machine Replication* - The application state should be replicated over all nodes in the consensus network to ensure that there is agreement on the confirmed transactions.
- *Modular Application-Consensus Design* - The type of consensus used by the system should be easily interchangeable without the need to modify the application itself.
- *Scalability* - The system should be scalable so that it is able to process a large volume of requests to simulate real world usage. The consensus network should also be scalable to handle more faults and malicious nodes for increased security.
- *Confirmed Transactions Written to Blockchain* - The system should integrate

a Blockchain ledger to write confirmed transactions to for later inspection and security.

Throughout the design and implementation of the application, these requirements will be used as a general target to achieve. In the Evaluation chapter, the success of the implementation will be measured against this set of requirements.

4.3 Design Principles and Reasoning

The principles driving the design of the application lie around the central topics of this paper.

- Using a consensus mechanism to validate client requests means that there is no single point of failure. Using multiple nodes to participate in the consensus provides extra security, malicious parties would need to obtain access to a majority of nodes to manipulate consensus results.
- Naturally, utilising a network of nodes to run the consensus leads to using State Machine Replication to run the application simultaneously across all nodes. Using SMR will ensure that the system state is kept consistent throughout all honest nodes in the network. This will result in only valid transactions being confirmed.
- Once confirmed, transactions will be written to a Blockchain ledger. This design choice makes transactions immutable once confirmed due to the property of Blockchain ledgers as discusses in the background chapter. The use of a Blockchain ledger also gives the system the properties of transparency and anonymity.

4.4 System Architecture

The architecture of the system should be based around the requirements specified previously. Consensus, SMR and Blockchain ledgers are the framework for this applications architecture. This system will be designed and modelled as a *permissioned* blockchain system, meaning consensus nodes will have to be validated before they can join the network,

4.4.1 Consensus Model

Byzantine Fault Tolerance consensus is the type of consensus that will be used, and in the background section of the project it has been established that there can be up to f byzantine nodes in a network of $3f + 1$ nodes. In the scenario of this system, the authority that is responsible for deployment and maintenance should be the responsible party for the consensus nodes. Client requests will all be transmitted to this consensus network, where all replicas will participate in a BFT-style consensus to validate and order the client requests. Malicious parties that want to tamper with the matching process will need to gain access to at least f nodes in the system. The value of f should be made as high as possible as this will increase the security and safety of the system.

An example model of the consensus process is shown in figure 4.2, where the red node is a byzantine node and the green nodes are functioning correctly. There are 4 nodes in the system and only 1 byzantine node so this network would still produce correct consensus results as per the BFT consensus property.

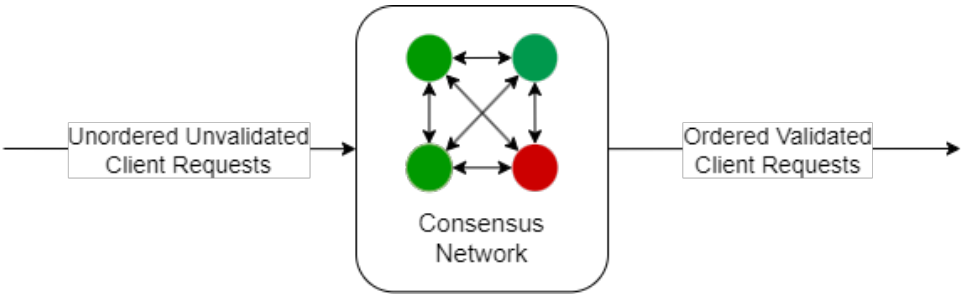

Figure 4.2: Consensus Model

4.4.2 State Machine Replication Model

Once all nodes have agreed on the validity and ordering of client requests, each request is executed sequentially by the application to update the system state. The application will be deterministic as stated in the requirements, to ensure that the same operation execution on all nodes with the same application state produces the same result state. The client matching algorithm will be a module of this application, which will output any valid client matches to create confirmed transactions. Trivially, any nodes who join the consensus network will need an up to date version of the application state to start executing new client requests. A model example is shown in figure 4.3.

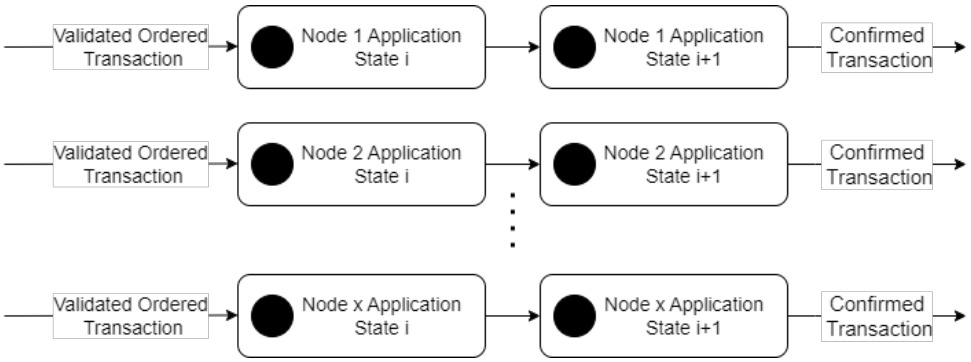

Figure 4.3: Consensus Model

4.4.3 Blockchain Ledger Model

Once transactions have been confirmed , they will be written into a Blockchain ledger. As discussed, this ensures that once transactions are confirmed, they become immutable and safe from tampering due to the properties of Blockchain ledgers discussed in the background chapter. Utilising a Blockchain ledger means all transactions are transparent and can be viewed by any interested third party. The Blockchain ledger model is shown in figure 4.4.

The original project plan was to integrate the system with the R3 Corda Blockchain to eliminate developing a new Blockchain component from scratch. However, due to the workload and time required to set this up, this step was left out in the implementation

Figure 4.4: Blockchain Model

4.4.4 Model Flow

Now that it has been established what architecture the system will use, a final model flow of the complete end to end system can be created. See figure 4.5 for a model reference on how each component of the system is used.

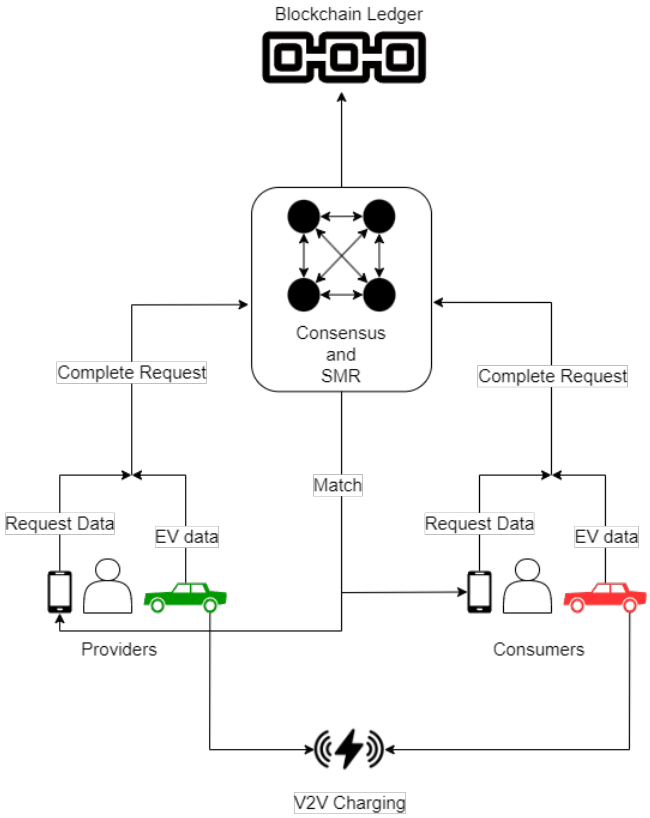

Figure 4.5: System Model

Chapter 5

System Design

Now that the model of the system has been laid out, the system design should naturally follow the structure of the model. As the implementation will be in the Java language, it is important that object oriented principles are adhered to whilst designing the system. The system must be designed on both client and server side so that complete functionality can be implemented following this design pattern.

5.1 Variables and Functions

In this section, the variables that will influence the matching process are defined. Consumers and providers have different requirements from which meeting point and partner client they are assigned to. Consumers want the match that will result in the lowest overall cost per unit of energy, and providers want the match that will result in the highest overall profit per unit of energy. One important model decision that is made here is that the area that the system operates in is represented as a coordinate system. For example, a consumer EV could be located at $(2,3)$, and a meeting point could be located at $(9,4)$. This simplification allows straightforward calculation of distances. From these derivations, UC and UP values can be calculated for each potential consumer and provider pair, while iterating through the various meeting points. (3.1) and (3.2) will be used in the implementation of the matching algorithm.

5.1.1 Unit Cost/Profit Functions

Separate functions can be created for both consumers and providers based off their requirements.

$$UC = \frac{C_C}{\alpha} \tag{5.1}$$

$$UP = \frac{R_P - C_P}{\alpha} \tag{5.2}$$

Where UC and UP are cost per unit of energy for consumers and profit per unit of energy for providers respectively. α is the total units of energy of being traded in the transaction. C_C and C_P are the cost functions for the consumer and provider respectively, and R_P is the revenue function for the provider. The maximisation of UC then UP (specifically in that order as consumers are prioritised over providers) is the goal of the matching algorithm in the system. Trivially, any match that will lead to a negative UP value should be discarded as it is a system failure if the provider makes a loss on the transaction. To model the cost and revenue functions for both provider and consumer, all factors that will influence the functions need to be considered.

5.1.2 Fee Function

The model for the transaction fee that the application will receive as its main source of revenue, can simply be modelled as:

$$F = r \times \alpha \tag{5.3}$$

where r is the constant that controls the size of the fee and $0 < r < 1$.

5.1.3 Consumer Functions

Starting with the consumer, the cost of travelling to the designated meeting point, the price for purchasing the energy from the provider and the fee for using the platform will affect the consumer cost function.

The purchase cost for the consumer for the energy is trivially:

$$EC = p_{SP} \times \alpha \tag{5.4}$$

where p_{SP} is the price per unit of energy set by the provider. The cost of travelling to the designated meeting point for the consumer can be modelled as:

$$TC_C = \sqrt{(x_C^2 - x_{MP}^2) + (y_C^2 - y_{MP}^2)} \times e_C \times p_C \tag{5.5}$$

where (x_C, y_C) and (x_{MP}, y_{MP}) are the coordinates of the consumer EV and designated meeting point respectively. e_C is the efficiency of the consumer EV and p_C is the price per unit that the original energy was bought for by the consumer.

The cost function for the consumer can be created as a sum of (3.3), (3.4) and (3.5):

$$C_C = EC + TC_C + F \tag{5.6}$$

5.1.4 Provider Functions

For the provider, the cost of travelling to the designated meeting point, the price that the energy was initially purchased, selling price and the transaction fee should be considered. The cost of travelling to the meeting point can be modelled in a similar manner to the consumer:

$$TC_P = \sqrt{(x_P^2 - x_{MP}^2) + (y_P^2 - y_{MP}^2)} \times e_P \times p_P \tag{5.7}$$

where (x_P, y_P) are the coordinates of the provider EV. e_P is the efficiency of the provider EV and p_P is the price per unit that the original energy was bought for by the provider. The model for the transaction fee is the same as (3.5). The cost function for the provider is trivially a sum of (3.7) and (3.5).

$$C_P = TC_P + F \tag{5.8}$$

The revenue function for the provider is just the revenue generated from the sale of the energy to the consumer:

$$R_P = p_{SP} \times \alpha \tag{5.9}$$

5.2 Client Side Design

System users require a client application to interface with the system. In this section the functionality and design of this client application is discussed.

5.2.1 Regular Client

The regular client will require the following functionality:

- Registering a new user. Client IDs must be unique.
- Logging in with registered Client ID.
- Linking EV to Client ID.
- Depositing/Withdrawing funds.
- Placing Buy/Sell Orders.
- Viewing Market Orders.

It is important to note that in a real world system deployment, sensitive operations such as logging in, registering and managing funds would have an additional layer of security or encryption to prevent fraudulent activity, but as this is not the priority for this project it has been left abstract. No data will be held on the client except for the logged in Client ID and associated vehicle information. For easy reference a skeleton class has

been given in the figure 5.1. These functions will each submit the corresponding type of request to the consensus network for ordering and validation.

5.2.2 Master Client

The authority responsible for running the system must have some elevated access and functionality. The master client has been designed for this purpose, in the case that orders or client information needs to be modified. The following functionality will be included in this client:

- Logging in with registered Master User ID.
- Managing User IDs. Clients who repeatedly misuse the system may have to be removed, new clients may be registered through the responsible authority.
- Modifying client funds. This functionality would be used in the situation clients need to be penalised for misusing the system or compensated for being involved in a failed transaction.
- Removing Buy/Sell Orders. If there are problems with specific orders, they may have to be removed from the system.
- Viewing Market Orders.

Like the regular client, only session information will be stored on the master client. The application state and logic will be entirely run on the nodes of the consensus and SMR network. These functions will each submit the corresponding type of request to the consensus network for ordering and validation. See figure 5.1 for a class skeleton.

5.2.3 Vehicle Information

Clients must be linked to an EV in order to place orders, as information is needed from the EV system such as battery state, location, telemetry and energy efficiency. A vehicle class is required to gather this information from the vehicle's onboard computer and transfer it to the client application. The EV state can then be accessed by the client when sending requests. See figure 5.1 for a skeleton.

5.2.4 Request Types

Client requests can be of two types:

- *Ordered Requests* - Requests that should be validated and ordered through consensus. This applies to requests that will alter the system state in some way, such as placing orders or managing funds.
- *Unordered Requests* - Requests that can skip consensus. This applies to requests that do not alter the system state, instead just read the system state, such as viewing the market and logging in.

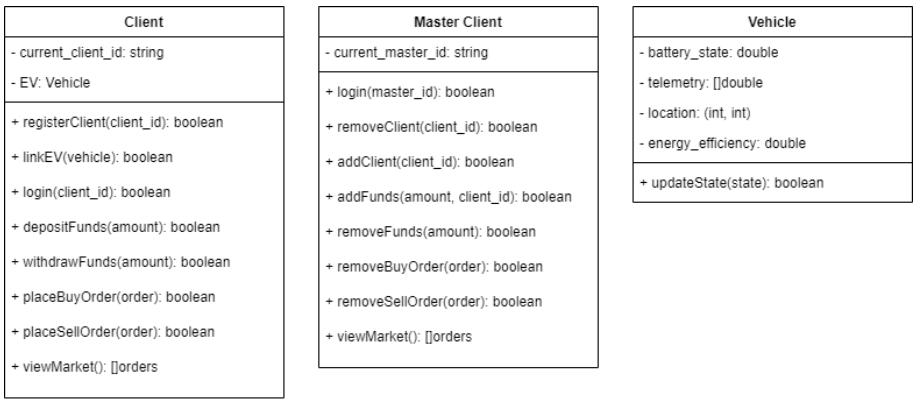

Figure 5.1: Client Class Skeleton

This differentiation allows requests that do not need ordering or validation to execute faster as they will bypass the consensus, and directly be passed to the SMR application. This also will improve the performance of the system as less time will be spent on executing consensus instances.

5.3 Consensus Design

As per the projects aim, three varying BFT consensus types will be used in the implementation. The consensus module must be kept abstract and separate in order to easily swap between consensus versions. As discussed in the background chapter, this was the main reason BFT_SMaRt was selected as the code base for this system, as it allows for SMR applications to be created without modifying the underlying consensus module.

5.3.1 Existing Design

For all three consensus versions, existing implementations were used with some modifications in order to tailor to the system requirements. All three versions share the same design pattern as they use BFT_SMaRt. See figure 5.2

All intra-network communication has been designed using a Netty framework and made abstract so point-to-point communication can be completed with simple function calls. Client requests are first delivered to the primary node which initiates a consensus instance for the request to be validated and ordered. In each consensus instance the low level algorithm and exchanged messages depend on what version of BFT is being used (see Background chapter for details). Once requests have been successfully ordered, they are delivered to the SMR application to be executed sequentially, and replies are sent back to the owner of the request.

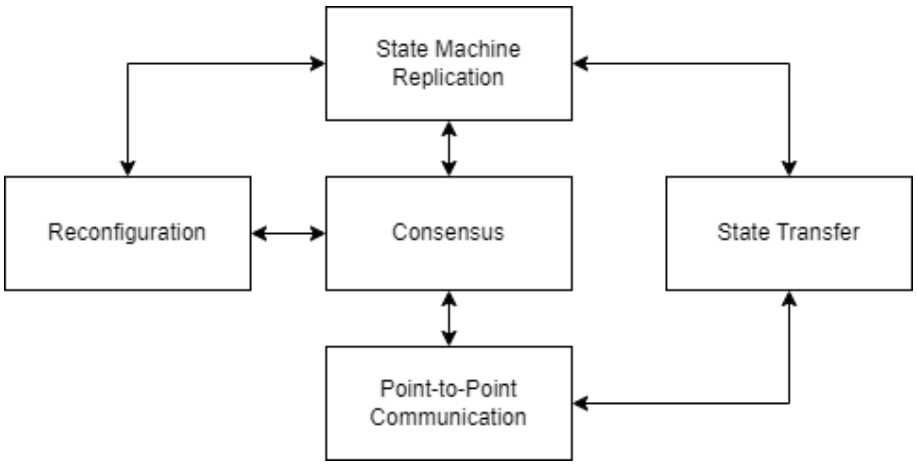

Figure 5.2: Consensus Design

5.3.2 Modifications

In the original implementations there was no validation of requests, the consensus had only ordering functionality. To simulate validation, the consensus module design was extended to include a validation module. This module ensures that requests received are from valid and trusted clients and the format matches the one that is required by the SMR application.

It is viable that the nodes running on the network change, so nodes must be able to send and receive states to catch up to the newest application state. The state transfer module design was modified to interface with the application to fetch and update the state when required, for example sending an updated state to a new node or installing the updated state in a new node.

5.4 Server Application

The server application is where the main logic will be located. All requests once validated and ordered will be passed to the application to be executed.

5.4.1 State Machine Replication

All nodes will run the server application in parallel, after applying consensus to requests for validation and ordering. The state of the application will be replicated across nodes as the order of operation execution will be consistent throughout all nodes in the network by consensus. However, to ensure the state is consistent the application must be deterministic, meaning an operation should have only one possible outcome. No random functions are used in the design to ensure that execution is always deterministic.

5.4.2 Class Architecture

Each instance of the server application will have its own database of client information such as what EV is associated with the client, available funds and placed orders. The

main class of the application will manage this database, fetch ordered and validated requests from the underlying consensus module and deliver replies back to clients. The order matching algorithm will be executed within this main class using the database already described. As the class structure is complex, a diagram has been shown in figure 5.3 for easy reference.

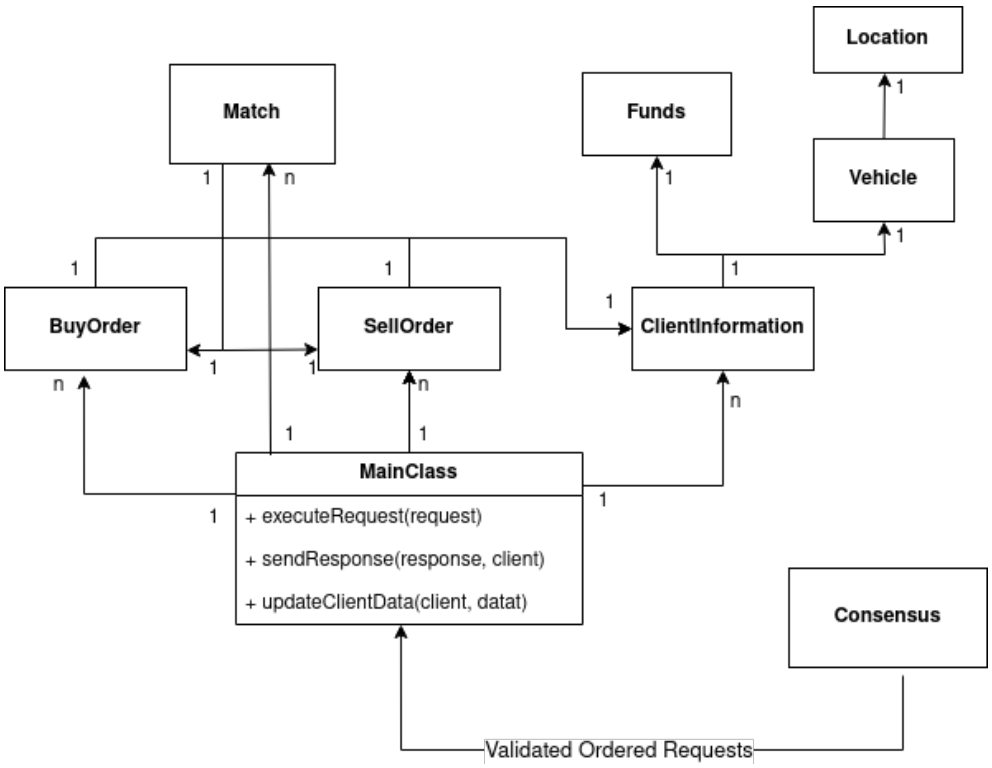

Figure 5.3: Class Model

5.4.3 Order Matching Algorithm

The algorithm to match buy and sell orders prioritises minimising the cost for consumers and maximising the profit for providers. The pseudo code for this design has been given for reference below. UC and UP are calculated by the equations derived in section 5.1. The variables required in these equations will be contained in client requests, where the client has submitted certain request parameters and certain request parameters are extracted from the linked EV.

Algorithm 1: Request Handling

```
Result: Order Matches
initialization;
while true do
    fetch next request r;
    if r.type is order then
        continue;
    else
        skip iteration;
    end
    extract order o from r;
    if o.type is sell then
        insert (orderID, o) into sellOrders;
        execute sellOrderHelper(o);
        set response;
    else
        insert (orderID, o) into buyOrders;
        execute buyOrderHelper(o);
        set response;
    end
    return response to client;
end
```

Algorithm 2: Sell Order Helper

```
Result: Order Matches
paremter o;
initialise bestMatch;
for bo in buyOrders do
    for mp in meetingPoints do
        calculate UC and UP for (o,bo,mp);
        if UC, UP better than bestMatch then
            update bestMatch to (o,bo,mp);
        else
            return nil;
        end
    end
end
if match found then
    remove bo from buyOrders;
    remove o from sellOrders;
    return bestMatch;
else
    continue;
end
```

Algorithm 3: Buy Order Helper

```
Result: Order Matches
paremter o;
initialise bestMatch;
for so in sellOrders do
    for mp in meetingPoints do
        calculate UC and UP for (o,so,mp);
        if UC, UP better than bestMatch then
            update bestMatch to (o,so,mp);
        else
            return nil;
        end
    end
end
if match found then
    remove so from sellOrders;
    remove o from buyOrders;
    return bestMatch;
else
    continue;
end
```

The complexity of this algorithm to process one request is $O(m \times n)$ where m is the number of buy/sell orders and n is the number of meeting points. Since n is a constant that can be controlled, the complexity can be reduced to $O(m)$.

5.4.4 Blockchain Ledger Design

The project objectives originally included integration with R3 Corda to have a fully functioning Blockchain system. However, due to time limitations, this was not possible in the final implementation. As a fallback, a dummy Blockchain scheme was designed to be used by confirmed transactions. This design uses a file to substitute a ledger, where each node would maintain its own copy of the file just as nodes on a real Blockchain network have a local copy of the ledger. Confirmed transactions are passed through a SHA-256 hash function and written to the ledger in blocks. The block size can be modified to adjust the rate of production of blocks. Every block except the genesis block will have the hash value of the previous block, just as a regular Blockchain does. To view the ledger the file can be read by other parties. New nodes joining the system can request a copy of this file or ledger to catch up the latest transactions in the Blockchain.

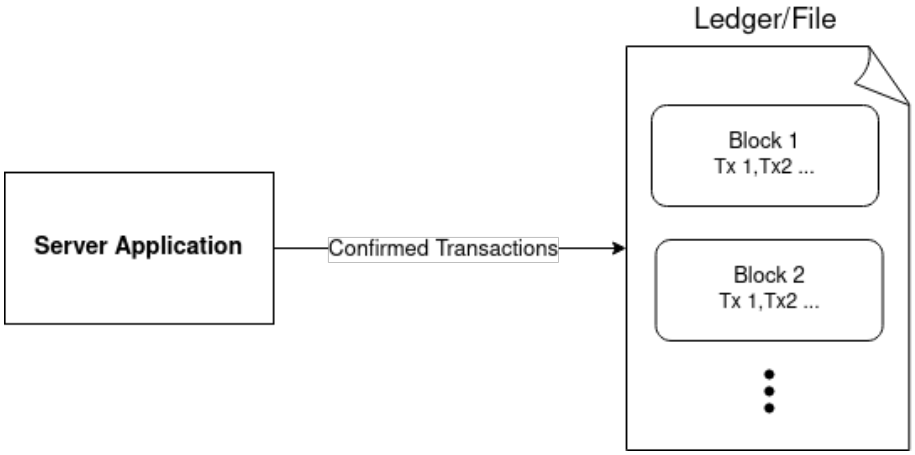

Figure 5.4: Blockchain Ledger Design

Chapter 6

Implementation

The entire system was implemented using the Java programming language, whilst following the design laid out in the Design chapter. The reasoning for this choice was that the BFT_SMaRt code base was based on Java, so it would make sense to use Java for the entire implementation for easy interfacing with BFT_SMaRt. In this section the implementation of each component in the system is discussed and how the functionality would be used in the real world.

6.1 Client Side Implementation

6.1.1 Regular Client

The regular client was implemented with two main components, an interactive client interface and a backend that communicates with the consensus network.

The interactive client provides an interface for users to use all functions of the system that are listed in section 5.2.1. The interface implementation is simple and easy to use, and gives instructions for each step. Once the user has logged in using the interactive interface, the client application will link all requests to the corresponding `client_id`. The extracted request data from user input is then passed to the backend. Once the request has successfully been completed, the response is displayed to the user and the user is free to submit another request. The code for this module is located in the `V2VInteractiveClient` class.

The backend client package receives unprocessed requests from the interactive interface and packages them into the structure that is required by the consensus nodes. All nodes require to be receive requests in Java byte array format, so the user input has to be parsed and converted and serialized into a byte array. One important implementation detail is that requests that do not change the state of the system such as viewing market data are packaged into unordered requests before being sent to the server. As discussed previously this allows these requests to bypass consensus to improve the performance of the system. The underlying Netty communication framework from the consensus module is extended to be used by the client, so the client can simply call functions from the communication module to send requests into the server. The backend also receives

responses from the server network and passes them to the server, deconstructs them from byte array format and passes them back the interface in a human readable format. The code for this module is located in the `V2VBackendClient` class.

6.1.2 Master Client

The master client was implemented with the same methodology as the regular client, with just the request types being modified to conform to the design discussed in section 5.2.2. The code for this client can be found in the `V2VInteractiveMasterClient` and `V2VBackendMasterClient` classes.

6.2 Consensus Implementation

6.2.1 Existing Libraries

Developing and implementing a BFT consensus system from scratch in the available time frame was not viable, as existing implementations of BFT consensus systems such as PBFT and UpRight contain upwards of 20k lines of code. For this reason it was decided to use the implemented BFT_SMaRt library for the consensus module.

The original BFT_SMaRt library consists of over 14k lines of Java code. Understanding this code base was a major part of development, as it was important to correctly interface with the consensus module. The implementations of Linear BFT and SBFT were based on the code base of BFT_SMaRt, so understanding the new libraries was less difficult after already having the knowledge on the baseline BFT_SMaRt.

6.2.2 Modifications

As discussed in section 5.3.2, the functionality of the consensus modules was changed during implementation.

In each implementation version, request validation was added inside the consensus module to ensure received requests were from trusted clients. This was implemented in the `Consensus` class in all three code bases by checking the `client_id` of the request with a hard coded white list of IDs. In a real world deployment, this would not be suitable as the clients can change, but for this projects purposes this implementation was sufficient for test purposes.

The state management module was modified to include sending and receiving the application state to allow new nodes in the system to receive the updated state of the application. This was implemented by serializing the application database into a Java byte array structure. When new nodes enter the network, they can request the latest version of the database from up to date nodes during the initialisation phase.

6.3 Server Application Implementation

The application that executes on each replica server is where the main logic of the system was implemented. The main class for this application is `V2VServer`. This class extends the `DefaultSingleRecoverable` class that is provided by BFT_SMaRt to make integration with the server application more streamlined.

6.3.1 State Machine Replication Implementation

As the consensus orders all incoming requests, SMR is trivially achieved if operation execution on the server application is always deterministic as long as there are no random properties included in the application. It was made sure during implementation to use only deterministic functions.

6.3.2 Request Handling Implementation

Validated and ordered requests are unpacked from Java byte array format. Depending on what request type is defined in the header of the request, the correct request function is called. This flow was implemented using a `switch` statement with all the request types, specified in the `V2VRequestType` class.

6.3.3 Database Implementation

The application stores information on orders and clients in a database. This database was implemented using Java `HashMap` and `HashSet` to link classes to others in a structured way, so that lookup, insertion and deletion could be performed easily using the associated IDs.

6.3.4 Algorithm Implementation

The order matching algorithm was implemented following the design in section 5.4.3. The translation from psuedocode to Java remained $O(m)$ complexity, where m is the amount of buy/sell orders. The database of orders is modified during the execution of this algorithm. The algorithm additionally checks that consumers have sufficient funds to complete the transaction it is confirmed. Upon confirmation, the respective client accounts are credited/debited and the confirmed transaction is sent back to the clients in Java byte array format.

6.3.5 Blockchain Ledger Implementation

The dummy ledger functionality was implemented by simply writing the hash of confirmed transactions into a file with the specified block size. The code for this functionality is located in the `LedgerWriter` class, and uses a Java `BufferedWriter` to write to a file.

6.4 Configuration Settings

Since there will be multiple nodes on the network running the same application, it must be specified what IP addresses and ports the nodes will be running on. Nodes can be added and removed by editing the `system.config` and `host.config` files. The codebase functions correctly with both local and external nodes. The payload size of requests will be modified during performance experimentation and can be adjusted in the `system.config` file. The block size for the block chain was also made to be adjustable in the same file.

6.5 Unit Testing

Individual functionality of each method in the client was tested by sending valid and invalid requests to the network and checking the response was correct. The unit testing included the following functionality checks:

- *Registering new users* - Registration with an ID that has already been registered results in failure. Unique IDs return successfully.
- *Logging in* - Invalid IDs were unable to log into the system, but registered IDs were able to login successfully.
- *Master Client Functionality* - The functionality of each method in the master client was correct, and it could modify user details are required.
- *Linking an EV* - Logged in users were able to link an EV to their ID.
- *Depositing/Withdrawing funds*
- *Placing orders* - Only logged in clients with linked EVs were able to place orders. The response was always displayed back to the user.
- *Viewing the Market* - Logged in clients were able to view the orders on the market at the current time.

The unit testing functionality of the server application included the following checks:

- *Database Management* - Insertion, deletion and updating records was always correct.
- *Order Matching* - If a valid match was found between provider, consumer and meeting point, it was always he match with the lowest UC and highest UP. When there was no match found, no confirmed transaction was outputted.
- *State Machine Replication* - The application state of all nodes hashed to the same value for after each operation execution.
- *Ledger Output* - Each replica produced a ledger file that was the exact same with the hash of confirmed transactions in the specified block sizes.

Chapter 7

Performance Evaluation

In this chapter, the performance of each version of BFT consensus is benchmarked by measuring throughput and latency whilst varying message size and network size.

7.1 Setup and Configuration

The aim of the performance experiments was to test the throughput and latency of the different versions of BFT consensus by simulating application use. Latency was measured as the average time taken for a response to be sent to a request from the client side, while throughput was the average number of operation executions per second the server applications achieved. The simulation of users submitting requests was implemented by creating Java threads for 10 different clients. Then for each client, requests were created through randomisation of request parameters. It was originally planned that the number of requests sent during the experiments would be high to simulate real world use, but the existing implementation of the SBFT consensus was not stable after for a large number of consensus rounds. To keep the amount of requests consistent between experiments, the requests were kept to an amount that would not break SBFT, which was discovered to be 200 requests per client, totalling 2000 requests being sent to the servers in total. To keep the requests consistent between experiments, a random seed was used to ensure that the request parameters would always be the exact same for each experiment run. In turn this ensures that the consensus would always receive the same requests to order and validate for each version of BFT consensus, and the application would execute the same operations in each experiment run.

The properties that were varied during the experiments were BFT consensus type, message size, and the number of nodes in the network. All server nodes were deployed locally on unique port numbers, the network sizes used were 4, 8, 16, 32 and 64 nodes. The message size was adjusted by padding requests with a Java byte array. The message sizes used were 8 bytes, 1 kilobyte and 10 kilobytes. The consensus type was trivially swapped by executing the associated consensus code base with the applications that were developed in this project. SBFT test runs were only in *Fast Path* mode as this is the main differentiation to Linear BFT.

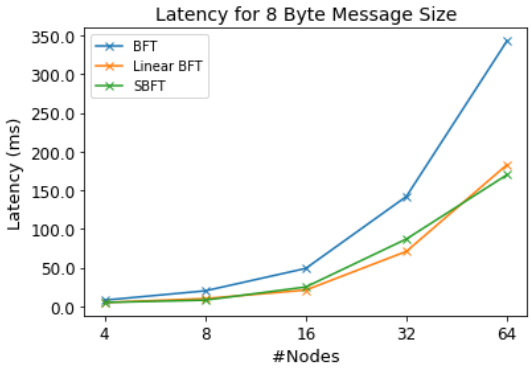

Figure 7.1: 8 Byte Message Latency

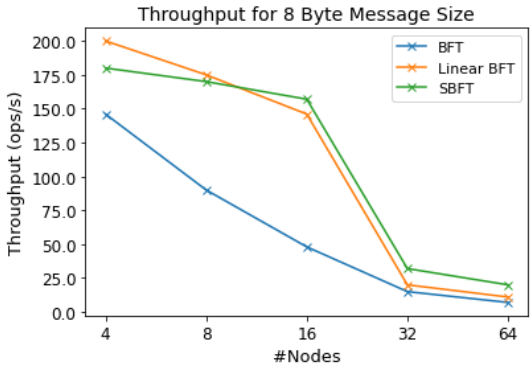

Figure 7.2: 8 Byte Message Throughput

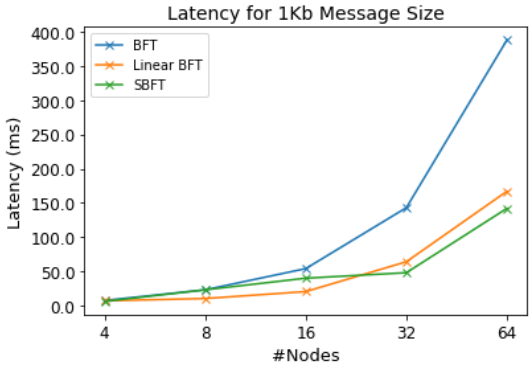

Figure 7.3: 1Kb Message Latency

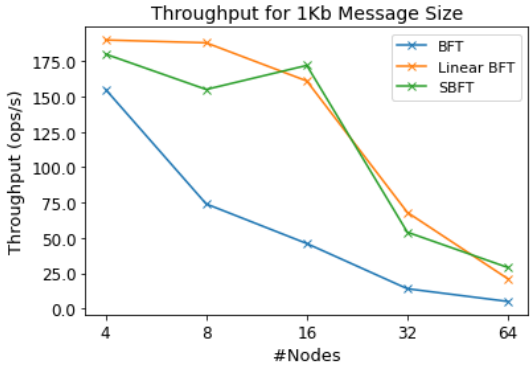

Figure 7.4: 1Kb Message Throughput

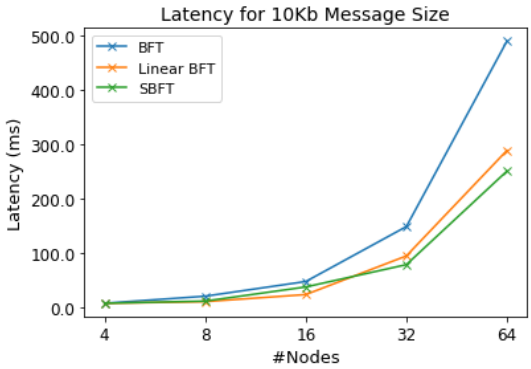

Figure 7.5: 10Kb Message Latency

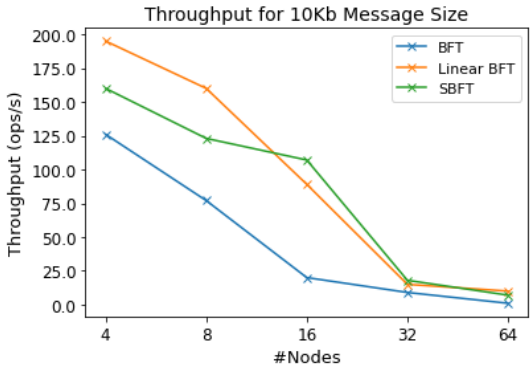

Figure 7.6: 10Kb Message Throughput

7.2 Results

See figures 7.1, 7.2, 7.3, 7.4, 7.5 and 7.6.

7.3 Analysis

7.3.1 Latency

The latency between using 8 byte messages and 1Kb messages did not have a large impact on any of the consensus variations [7.1 and 7.3]. However using an extremely large message size did have a noticeable performance loss [7.5]. For some network sizes, this value almost doubles. This could be explained by the fact that larger message sizes means the underlying Netty communication framework for the consensus becomes a bottleneck as too much data is being sent in each message. As the number of nodes in the network increases, Linear BFT and SBFT perform better. This is to be expected as the exploitation of TSS allows for $O(n^2)$ message exchange complexity, Since there are less messages being exchanged between nodes, there is less bottleneck in the system and latency will be lower, Strangely however, the performance of Linear BFT and SBFT is rather similar, when ideally SBFT should outperform Linear BFT due to its *Fast Path* feature where it skips the *Prepare* step of the consensus algorithm. When the network size is relatively small, the performance difference in latency between the consensus algorithms is negligible as only a relatively small amount of messages would need to be exchanged in any consensus variation.

7.3.2 Throughput

The throughput values are somewhat stable between message sizes [7.2, 7.4 and 7.6]. The explanation for this could possibly be that once the consensus has received a large message, unpacking and processing time and time for consensus to be achieved is negligible compared to the time spend in sending the large message through the network, which affects latency but not throughput. There appears to be an optimal range in which both SBFT and Linear BFT perform significantly better than regular BFT on all message sizes. This range is around 8-32 nodes where the gap between the throughput of regular BFT and both Linear BFT and SBFT is large. When the network size is relatively low, the throughput performance is stable between all versions of consensus as the number of messages exchanged is low, leading to faster consensus decisions and in turn operation executions happen more frequently on the server application. In the 8-32 node range mentioned, the conditions are favourable for both Linear BFT and SBFT to exploit their linear message complexity property to achieve consensus with less message exchanges, However, for larger networks, the exchange of messages becomes the influential factor regardless of message complexity, leading to a lower throughput for all three versions of consensus. It should be noted again that SBFT did not outperform Linear BFT, even though this was expected due to SBFT being able the *Prepare* step while running the *Fast Path*.

7.3.3 Comparison to Past Work

These findings are consistent with previous work such as the mentioned papers [28] and [31] where regular BFT [9] is one of the slowest performing consensus algorithms. The performance tests in the respective papers for both SBFT [22] and Linear BFT [27] share similar results where they both outperform regular BFT easily. However, the

results obtained by SBFT in this project are not as impressive as the ones obtained in the SBFT paper, where the performance of SBFT in the *Fast Path* is up to 3 times better.

7.3.4 Summary

It is clear from the performance tests that Linear BFT and SBFT outperform regular BFT. There is a range between 8-32 nodes on the network that has conditions that are most favourable for SBFT and Linear BFT, where high throughput can be achieved whilst maintaining low latency from the client side. One unexplained discovery was the similarity in performance between SBFT and Linear BFT, SBFT running with the *Fast Path* property was expected to outperform Linear BFT, but failed to do so. This may be a problem with the implementation of SBFT where the code design may not be as efficient as the Linear BFT code.

Chapter 8

Discussions

8.1 Achievements

In this project, a Blockchain-based V2V Energy Trading system that uses BFT style consensus was successfully modelled, designed and implemented. Looking back at the requirements specified for the system in section 4.2, each requirement was fulfilled by the end of the final implementation.

- *Basic Request Functionality* - Clients can use all functionality that was originally designed.
- *Deterministic Buy/Sell Request Matching Algorithm* - The algorithm to match orders successfully passed the unit test for determinism.
- *Request Validation and Ordering* - The system integrates BFT consensus to order and validate requests
- *State Machine Replication* - The nodes on the system network converge to the same state for the same operation executions.
- *Modular Application-Consensus Design* - The version of BFT consensus is easily able to be interchanged without modifying the application code base.
- *Scalability* - The network was able to scale to 64 nodes to allow for up to 21 corrupt nodes for increased security.
- *Confirmed Transactions Written to Blockchain* - The hashed transactions were written to a file for ledger functionality simulation.

8.2 Limitations

There are certain limitations to the work completed in this project. The SBFT implementation used was unstable for a large amount of requests, which limited the number of requests that could be sent for testing both the stable Linear BFT and regular BFT.

Additionally, the affect of having byzantine nodes was not investigated as the SBFT implementation did also not include a switch from the *Fast Path* to the *Slow Path*, meaning the algorithm would get stuck when nodes become byzantine. To keep tests consistent between all versions of BFT, it was decided to avoid this experiment. The original project plan included integrating R3 Corda into the system. but due to time frame limitations this could not be achieved. Instead, dummy Blockchain ledger functionality was used by writing hashed transactions to a file. The experiments were all run locally, instead of deploying the nodes on the cloud at various locations to better simulate a real distributes system.

8.3 Future Work

The future work for this project is naturally based around the current limitations. Using an implementation of SBFT with complete functionality would allow for additional experiments to further investigate what effects each version of the BFT consensus have on application performance. Furthermore, new versions of BFT consensus such as HotStuff [1] and Tendermint [5] could also be used with the application to see if there is any improvements compared to the currently used consensus types. Running the consensus network on the cloud across different physical locations would better simulate a real world deployment of the system, where server replicas are spread about an area for security. This would give more realistic results for throughput and latency experiments. Perhaps the main work that could be completed to extend this project is integrating with a developed Blockchain system such as R3 Corda for additional security and functionality. This would also allow for more realistic performance test scenarios.

Chapter 9

Conclusion

Comprehensive model, design and implementation of a Blockchain-based V2V Energy Sharing system using BFT consensus have been discussed and analysed in this project. The final implementation of the interfacing application is robust and reliable, and provides the functionality a real world deployment of this system would use. The application was designed to be modular so that different versions of BFT consensus based on the BFT_SMaRt code base could be used to compare and test the performance results. A ledger prototype was implemented in place of integration with an existing Blockchain system, which perhaps is the main limitation of this project especially compared to previous work. The system was evaluated through simulation experiments, which have shown that it can achieve high levels of throughput and scalability.

There has been previous work completed in this field, but the unique contribution of this project was the use of Linear BFT and SBFT with a V2V Energy Sharing system. No other published work has discussed or tested these types of BFT consensus. It was discovered that these two BFT consensus versions were very similar in performance tests for throughput and latency, but both performed significantly better than the regular BFT consensus algorithm.

Overall, this dissertation contributes to the growing body of literature on Blockchain-based V2V energy trading, and provides valuable insights into the potential of this technology to transform the energy sector. With further research and development, it is hoped that Blockchain-based V2V energy trading systems can be successfully deployed on a large scale, leading to a more sustainable and decentralized energy future.

Bibliography

[1] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.

[2] Alysson Bessani, João Sousa, and Eduardo Alchieri. State machine replication for the masses with bft-smart. pages 355–362, 06 2014.

[3] Dan Boneh, Marten Drijvers, and Gregory Neven. Bls signatures. *IACR Cryptology ePrint Archive*, 2018:1–26, 2018.

[4] Dan Boneh, David M Freeman, and Jonathan Katz. Distributed key generation in the wild. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 266–282. Springer, 2003.

[5] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[6] Vitalik Buterin. Ethereum white paper. https://ethereum.org/en/whitepaper/, 2014. Accessed on April 24, 2023.

[7] Vitalik Buterin. The next frontier in cryptocurrency: Scalability. In *ACM SIGSAC Meeting on Computer and Communications Security*, pages 123–124. ACM, 2014.

[8] Christian Cachin and Marko Vukolic. Architecture of the hyperledger blockchain fabric. *Proceedings of Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[10] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology Proceedings of Crypto*, pages 199–203. Springer, 1982.

[11] Konstantinos Christidis and Michael Devetsikiotis. Understanding the blockchain technology behind bitcoin: An energy perspective. *IEEE Transactions on Smart Grid*, 9(3):2169–2177, 2018.

[12] Allen Clement, Edmund L Wong, Lorenzo Alvisi, and Michael Dahlin. State-machine replication scalability made simple. In *Proceedings of the 23rd ACM symposium on Operating systems principles*, pages 153–168, 2009.

[13] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. Bitcoin: Under the hood. *Communications of the ACM*, 58(9):104–113, 2015.

[14] John E. Hopcroft and Jeffrey D. Ullman. Introduction to automata theory, languages, and computation. *Addison-Wesley*, pages 106–109, 1979.

[15] Peter Joseph, Elangovan Devaraj, and Arunkumar Gopal. An overview of wireless charging and v2g integration of electric vehicles using renewable energy for sustainable transportation. *IET Power Electronics*, 12, 04 2019.

[16] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, 2013.

[17] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[18] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Proceedings of the 4th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 382–401. ACM, 1982.

[19] Xiaolong Li, Xiaoliang Liu, Wei Chen, Feng Xia, Hongbin Sun, and Yonggang Wen. Internet of electric vehicles: architecture, applications and technologies. *IEEE Internet of Things Journal*, 4(6):2012–2023, 2017.

[20] S. Mazloom, B. Zhang, G. Pau, and H. Li. Cityflow: A vehicle-to-grid energy management platform for smart cities. *IEEE Transactions on Smart Grid*, 11(2):1569–1579, 2020.

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[22] Rafael Pass and Elaine Shi. Sbft: a scalable and decentralized trust infrastructure for blockchains. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 429–447. ACM, 2017.

[23] J. Postel. Transmission Control Protocol. In *RFC 793*, 1981.

[24] R3. Corda: An introduction. *R3 White Paper*, 2016.

[25] Melanie Swan. Blockchain: blueprint for a new economy. *O'Reilly Media, Inc.*, 2015.

[26] Ethan C Tse, Li Zhou, and Yan Li. Blockchain reliability: A perspective. *IEEE Access*, 6:20238–20253, 2018.

[27] Yin Yang. Linbft: Linear-communication byzantine fault tolerance for public blockchains. *CoRR*, abs/1807.01829, 2018.

[28] Mingrui Yin, Yu Yang, Wenjie Li, and Xiang Huang. Consensus mechanism for blockchain-enabled vehicle-to-vehicle energy trading in the internet of electric vehicles. *IEEE Access*, 9:107894–107905, 2021.

[29] Weisheng Yue, Hongfei Fan, Zheng Zhang, Yajun Xu, Kui Zhang, and Siyuan Cheng. An efficient vehicle-to-vehicle (v2v) energy sharing framework. *IEEE Transactions on Industrial Informatics*, 17(4):2813–2823, 2021.

[30] Maryam Zahedi, Mehrdad Nouri Moghaddam, Naser Movahhedinia, Masoud Javadi, Behnam Mohammadi-Ivatloo, and Jamshid Aghaei. echarge: A blockchain-

based platform for peer-to-peer electric vehicle charging. *Sustainable Cities and Society*, 42:251–260, 2018.

[31] Jinyu Zhang, Anhong Zhou, Bin Hu, and Jianguo Yang. A fast and secured vehicle-to-vehicle energy trading based on blockchain consensus in the internet of electric vehicles. *IEEE Access*, 7:103902–103913, 2019.

[32] Eric P Zheng. *Blockchain and healthcare strategy guide*. CRC Press, 2017.