

Peer-to-Peer Energy Trading System using IoT and a Low-Computation Blockchain Network

Submitted in fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

of Rhodes University

Tyron Ncube

Grahamstown, South Africa
June 2021

Abstract

The use of renewable energy is increasing every year as it is seen as a viable and sustainable long-term alternative to fossil-based sources of power. Emerging technologies are being merged with existing renewable energy systems to address some of the challenges associated with renewable energy, such as reliability and limited storage facilities for the generated energy. The Internet of Things (IoT) has made it possible for consumers to make money by selling off excess energy back to the utility company through smart grids that allow bi-directional communication between the consumer and the utility company. The major drawback of this is that the utility company still plays a central role in this setup as they are the only buyer of this excess energy generated from renewable energy sources.

This research intends to use blockchain technology by leveraging its decentralized architecture to enable other individuals to be able to purchase this excess energy. Blockchain technology is first explained in detail, and its main features, such as consensus mechanisms, are examined. This evaluation of blockchain technology gives rise to some design questions that are taken into consideration to create a low-energy, low-computation Ethereum-based blockchain network that is the foundation for a peer-to-peer energy trading system. The peer-to-peer energy trading system makes use of smart meters to collect data about energy usage and gives users a web-based interface where they can transact with each other. A smart contract is also designed to facilitate payments for transactions. Lastly, the system is tested by carrying out transactions and transferring energy from one node in the system to another.

Keywords : Blockchain, Internet of Things (IoT), Renewable energy, energy trading systems

Acknowledgements

This project would not have been possible without the help and cooperation of many. Firstly, gratitude goes out to my supervisor Prof Nomusa Dlodlo for the guidance she provided throughout the duration of this research. This would not have been possible without her knowledge of the research area. Prof Alfredo Terzoli also deserves an honourable mention for the help he provided and the tough questions he always asked. Thanks also goes out to the colleagues I shared a lab with, especially Mulalo for listening to my brainstorming and offering ideas when I was stuck. I would also like to thank Mr John Gillam for assisting with funding opportunities. Most of all I would like to thank God for making this possible during these unprecedented times.

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from the Henderson Scholarship fund and the Rhodes University Postgraduate Funding office. The author acknowledges that the opinions, findings and conclusions or recommendations expressed here are those of the author and that none of the above mentioned sponsors accept liability whatsoever in this regard.

TABLE OF CONTENTS

List of Figures	iv
List of Tables	viii
1 Introduction	1
1.1 Background of the Research	1
1.2 Problem Statement	1
1.3 Aim of the Research	2
1.4 Research Objectives	2
1.5 Justification	2
1.6 Thesis Outline	3
1.7 Scope of the Study	4
2 Background	5
2.1 Blockchain Overview	5
2.2 Internet of Things Overview	23
2.3 Related Work	27
3 System Architecture	34
3.1 Design Approach	34
3.2 High-level System Design	35
3.3 Peer-to-peer Energy Trading System Use-Case	36
3.4 Peer to Peer Architecture	38
3.5 Software Architecture	39
3.6 Hardware Architecture	47

3.7	Data Transfer Protocols	49
3.8	Conclusion	51
4	Private Blockchain Network	52
4.1	Ethereum Virtual Machine	52
4.2	Ethereum Node	53
4.3	Ethereum Client	54
4.4	Setting up an Ethereum Node	56
4.5	Setting up a Private Ethereum Network	58
4.6	Genesis File	62
4.7	Starting the Blockchain Network	64
4.8	Geth Console	67
4.9	Sealer Nodes	71
4.10	Block Validation on a Clique-based Ethereum Network	72
4.11	Conclusion	74
5	Smart Contract Implementation	76
5.1	Ethereum Smart Contract Execution	76
5.2	Smart Contract for the Proposed System	77
5.3	Alternative Approach to the Smart Contract Design	81
5.4	Smart Contract Functionality Testing	82
5.5	Deploying the Smart Contract	87
5.6	Client-side Application	88
5.7	Client-side Application Access Control	93
5.8	Conclusion	93
6	Hardware Implementation	94
6.1	Hardware Components Considerations	94
6.2	Sonoff POW R2 Smart Meter	97
6.3	Sonoff POW R2 Firmware	99
6.4	Tasmota Firmware	100

6.5	Storing Smart Meter Data to a Database	104
6.6	Energy Transfer	107
6.7	Measuring Energy Transferred	109
6.8	Checking Transaction Status	111
6.9	Conclusion	111
7	System Testing	112
7.1	Test Set-up	112
7.2	Adding New Listing	115
7.3	Making a Purchase	117
7.4	PoA-based Blockchain Resource Usage	122
7.5	Conclusion	125
8	Conclusion	126
8.1	Achieved Objectives	126
8.2	Future Work	128
8.3	Summary	128

List of Figures

2.1	Structure of a block in a blockchain[7]	6
2.2	Example of a proof of work puzzle	9
2.3	Seven layer blockchain architecture [23]	17
2.4	Three layer blockchain architecture [26]	18
2.5	Smart contract use-case diagram	22
2.6	IoT Architecture [42]	24
3.1	High-level system design of the peer-to-peer energy trading system	35
3.2	Use-case diagram of peer-to-peer energy trading system	37
3.3	Illustration of peer to peer architecture	38
3.4	Software architecture of the peer-to-peer energy trading system	40
3.5	Hardware Architecture of the proposed system	48
3.6	MQTT Architecture	50
4.1	Features of an Ethereum client[102]	54
4.2	Creating working directories	56
4.3	Creating a new Ethereum account	57
4.4	Starting up Puppeth and creating a new genesis file	59
4.5	Selecting the consensus engine using Puppeth	60
4.6	The generated genesis file	62
4.7	Initializing the genesis file	64
4.8	Starting the Geth console	65
4.9	eth.log file after the Geth console is started	67
4.10	Unlocking account and checking account balance	68

4.11	Checking for other nodes on the network	68
4.12	enode of the first node	69
4.13	The second node joining the network using the enode of the first node . .	69
4.14	First node checking for other nodes on the network	70
4.15	Account balance of the second node	70
4.16	Transaction to send Ether from the first node to the second node	70
4.17	New account balance of the second node	71
4.18	Command to propose a new sealer node	72
4.19	Command to vote out a sealer node	72
4.20	Command to cancel a proposal that has already been made	72
4.21	Group of sealer nodes at time t1 where N=8	73
4.22	Group of sealer nodes at time t2	74
5.1	Smart contract code for the listElectricity function	77
5.2	ListingAdded event code of the smart contract	78
5.3	purchaseElectricity function of the smart contract	79
5.4	Smart contract code for the paySeller function	79
5.5	The purchaseFailed function code in the smart contract	80
5.6	Smart contract test questions	82
5.7	Test code to check smart contract deployment	83
5.8	Results after running test program	83
5.9	Test code to retrieve electricity listings	84
5.10	Results after running test program to retrieve listings	84
5.11	Test code to check the accuracy of the data retrieved	85
5.12	Test code to check if the seller is paid after a transaction	86
5.13	Snippet of the Truffle configuration file showing network settings	87
5.14	Results after successfully deploying the smart contract to the blockchain	88
5.15	Code for the listElectricity function of the client-side application	89
5.16	Screenshot showing the parameters required to add a network to Metamask	90
5.17	Code for checking if the browser supports Ethereum distributed applications	91

5.18	Screenshot of user interface of the web application	92
6.1	Power bank used in the peer-to-peer energy trading system	96
6.2	Components of Sonoff POW R2 smart meter	97
6.3	Top view of Sonoff POW R2 smart meter	98
6.4	Tasmota web interface home page	102
6.5	Tasmota MQTT configuration	103
6.6	Tasmota time period configuration	104
6.7	Data log on the Tasmota console	104
6.8	Screenshot of data sent by the smart meter	105
6.9	Node-Red flow for subscribing to smart meter MQTT topic	105
6.10	Console of Node-Red client subscribed to smart meter MQTT topic . . .	106
6.11	Node-Red flow for saving data to a database	107
6.12	Multiple Node-Red flows for saving data from multiple smart meters to a database	108
6.13	Connection of components on a single node	109
6.14	Connection of multiple relays to the same ground and 5V terminals . . .	110
7.1	Hardware set-up of three nodes	114
7.2	Adding new listing through the web-based application	115
7.3	Transaction fee for adding a new listing	116
7.4	Browser notification from Metamask after the transaction is added to the blockchain	116
7.5	Web-based application after the listing is added	117
7.6	Database table showing the energy balances of all three nodes	118
7.7	Web application after successful transaction	118
7.8	Metamask screenshot showing transaction details	119
7.9	Database table showing energy balances of all the nodes before the start of a transaction	120
7.10	Database table showing energy balances 6 minutes after transaction started	121

7.11 Metamask screenshot showing the amount paid to the seller and transaction cost	121
7.12 Web-based application after conclusion of the transaction	122
7.13 Resource usage on Node 3	123
7.14 Resource usage on a sealer node	124
7.15 Resource usage on node 1 during transfer of energy	124

List of Tables

6.1	Sonoff POW R2 technical specifications [117]	99
6.2	Sonoff POW R2 wireless specifications [117]	99
7.1	Raspberry Pi 3 Model B+ Specifications [123]	112

Chapter 1

Introduction

1.1 Background of the Research

Power generation is the backbone of development in any modern society and most initiatives meant to assist developing communities require a reliable and affordable source of electrical power. Currently fossil-based fuels account for a large proportion of all the energy used by consumers [1] [2] but the use of renewable energy is increasing every year as it is seen as a viable and sustainable long term alternative to the fossil-based sources of power.

Various emerging technologies are continuously being merged with existing renewable energy systems to address some of the challenges associated with renewable energy such as reliability and limited storage. Smart grids have made it possible to have two-way communication between energy producers and consumers. When integrated with renewable energy systems, smart grids have made it possible for consumers to sell back excess energy from their renewable energy systems to the utility companies. Hybrid energy systems that use multiple renewable energy sources have also become popular due to technologies like the internet of things (IoT) making them more reliable.

1.2 Problem Statement

In most of the above-mentioned smart grid systems, the utility company still plays a significant role such as being the main supplier of energy as well as being the sole buyer of the excess energy produced by its consumers. This results in a centralized monopoly where the utility company is free to set terms and conditions that benefit it more than

the consumers.

1.3 Aim of the Research

The aim of the research is to eliminate the monopoly that the utility company has over the purchasing of excess energy produced by consumers by coming up with a system that allows other participants to be able to purchase it as well. The proposed system leverages the decentralized architecture of blockchain and uses IoT to collect data from the renewable energy systems to make the system more efficient.

1.4 Research Objectives

The objectives of this research can be classified into a primary objective as well as secondary objective. The primary objective of this study is detailed below:

- To create a prototype for a blockchain-based peer-to-peer energy trading system where users who produce renewable energy can buy and sell the excess energy amongst each other without the need for a central authority.

The secondary objectives of this study are listed below:

- Using IoT to collect information from the existing renewable energy systems and using this information to make the peer-to-peer energy trading system more efficient by presenting this information to the user to help them make better informed decisions.
- To come up with a blockchain configuration that does not require intensive computer resources but still maintains the blockchain properties that make it suitable for the peer-to-peer energy trading system.

1.5 Justification

According to various reports, at least half of the people in sub-Saharan Africa have no access to electricity due to low power generation but more and more settlements are being electrified every year [3] [4]. This increase in the number of settlements that are being

electrified does not, however, coincide with an increase in the power generation capacities of the different countries. This means that as more households are added to the main power grid, it puts a strain on the grid as they all have to share an already inadequate power supply.

The proposed system offers a set up where settlements can be equipped with renewable energy systems such as solar or wind turbines and households within a settlement can be connected to each other to create a small smart grid where the households can trade excess energy with each other. This approach reduces the need for the settlement to rely on electricity from the main grid, which can instead be used as a backup source. It also increases the number of households that use renewable energy as their main source of electricity instead of fossil-based sources.

The use of blockchain technology is ideal to address the problems identified for a number of reasons. It shares a similar architecture with the proposed system in that they are both peer-to-peer. This helps avoid a situation where the hardware is configured in a peer-to-peer architecture but the software is centralized. Blockchain also offers a payment medium for the transactions in the proposed system which eliminates the need for a separate payment platform. The transparency offered by blockchain is also ideal for a system where the role of the trusted third-party in the form of the regulator is significantly reduced.

1.6 Thesis Outline

This document is organized into nine chapters and some appendices that contain supporting material. The rest of the chapters are organized as follows:

- Chapter 2 contains an overview of the two main technologies that are used in this research, namely blockchain and the internet of things. The chapter also evaluates similar studies that have been carried out.
- Chapter 3 proposes the architectures for both the hardware and software components of the system to be developed as well as the design approach to take in the implementation of the proposed solution.

-
- Chapter 4 gives a more technical background on how Ethereum handles transactions. It also details the steps to take to create an Ethereum-based private blockchain network.
 - Chapter 5 involves the design, coding and implementation of the smart contract for the peer-to-peer energy trading system.
 - Chapter 6 gives a detailed account of the hardware configuration of the proposed system and how it interacts with the software.
 - Chapter 7 carries out various tests on the peer-to-peer energy trading system to ascertain how it performs under different conditions.
 - Chapter 8 concludes the research by highlighting how the results obtained compare with the initial objectives of the research and suggests future work.

1.7 Scope of the Study

This research focuses on the development of a peer-to-peer energy trading system in order to address the issues identified in the problem statement. The research, however, focuses on a narrow scope. Firstly, the study is limited to peers within close physical proximity to each other. This is to minimize the energy loss during transfer from a seller to a buyer. The other assumption is that all the peers in the system already have renewable energy systems as the study builds on already existing renewable energy systems.

Chapter 2

Background

This chapter gives a detailed overview of blockchain and IoT technologies by defining what they are and how they function. The evaluation of some similar studies that have been carried out by other scholars is also carried out in this chapter and differentiates this research from those studies.

2.1 Blockchain Overview

This subsection defines a blockchain and details how the technology works. The technical specifications of blockchain technology are also assessed.

2.1.1 Introduction

A blockchain is a distributed digital ledger that contains blocks of transactions [5]. A block is a file within a blockchain that stores transactions that have not yet been recorded on any prior blocks. Thus, a block is like a page of a ledger or record book. A blockchain can also be defined as a back-linked ledger containing blocks of transactions whose sequence is agreed on by an evolving set of nodes [6]. The main goal of blockchain technology is to enable transactions between two mutually distrusting parties without a trusted third-party, and it does this by distributing trust among different parties who get an economic incentive for acting honestly. If the blockchain's distributed parties cannot agree on the blockchain's correct state or if there are conflicting versions of the blockchain, the version with the longest chain, i.e., the one with the most subsequent blocks after the conflict, is the one taken to be valid.

2.1.2 Blockchain Structure

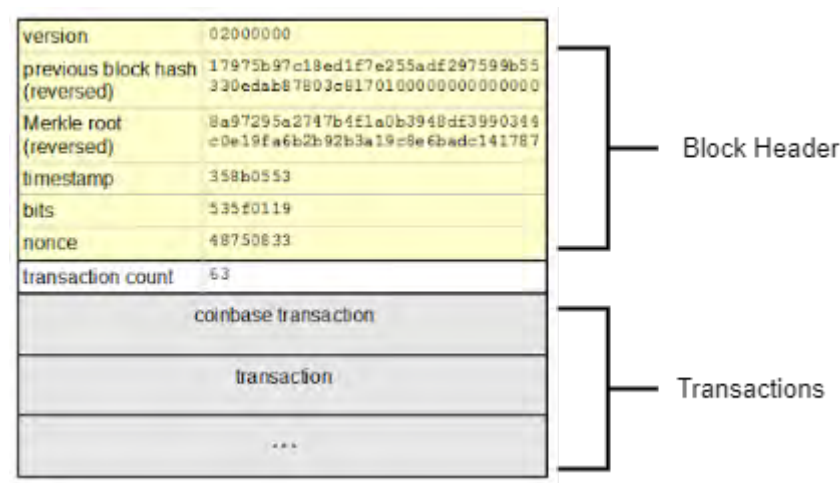


Figure 2.1: Structure of a block in a blockchain[7]

A block in a blockchain contains a block header as well as a list of all the transactions in that block. The block header contains the block number, the previous block's hash value, the block's timestamp, the nonce value, and the block's size as illustrated in Figure 2.1. The blocks in a blockchain are uniquely identified using a hash algorithm found on the header of each block. Each block contains the hash of the block immediately preceding it, and that hash is used to compute its own hash value. Therefore, these hash values link all the blocks in a blockchain back to the first block. When the block preceding a particular block changes, this then changes the hash of that block and every other block that came after it, which means the hashes of all those blocks will have to be recalculated. The calculation of these hashes is a computationally-intensive process whose cost would outweigh any benefits that can be derived from the changes to the blockchain. This acts as a barrier against maliciously adding a block in the chain and also ensures the immutability of transactions in a blockchain by guarding against changes to existing blocks [8].

2.1.3 Genesis Block

The genesis block, which is also known as the genesis file, is the first block in any blockchain-based protocol [9]. It is the foundation upon which subsequent blocks are added to the blockchain as it contains rules that govern how the blocks are added to the

blockchain. Rules that govern the addition of new blocks include the method that is used to validate new blocks as well as the method for selecting the node that does this validation. The genesis block comes pre-configured with the system as it is the blockchain's initial state, and every node that joins the network has to agree to this initial state. The state of a blockchain is the information about the blockchain such as the number of blocks it contains at that given point in time as well as the sequence of the blocks. Any successful transaction that is added to the blockchain changes the blockchain state.

2.1.4 Transactions in a Blockchain

A transaction in a blockchain is any action that changes the state of the blockchain. Different blockchains have different actions that can change the state of the blockchain. With Bitcoin, for example, the only transactions involve the sending and receiving of cryptocurrency as well mining but with other blockchains such as Ethereum, a transaction may also involve running a smart contract on the blockchain. When a transaction is initiated in a blockchain by a user, it must be first signed using the user's private key in order to authenticate its origin. The private key is what gives a user the ownership of an address in a blockchain and a user cannot access the cryptocurrency in an address without the private key. After signing, the transaction is then broadcast to all the other nodes in the network before it is collected with other transactions and grouped into a candidate block.

Before a candidate block is added to the blockchain, it must be validated first through a process known as mining. The nature of the mining process depends on the method used to reach consensus on the network [10]. In some networks such as Bitcoin, it involves finding a number that produces a hash value with a specified number of leading zeroes. After the mining process is complete, other nodes can check that the block is valid and that it contains the correct hash value of the previous block. If all the details are correct, then the block is added to the blockchain, but if there are any errors, then the block is discarded [11].

2.1.5 Consensus Mechanisms

A consensus mechanism can be defined as the method used to agree on the network's correct state i.e. the correct sequence of blocks in a blockchain system [12]. Consensus mechanisms are used to determine how new blocks are published, which node gets to publish the new block and how to solve any conflicts that may arise from the process of publishing a block, such as what happens when two or more nodes publish a block at approximately the same time [7]. An example of a conflict resolution rule when two nodes publish blocks at the same time, is to temporarily create two separate chains for the two different blocks and continue with both chains until one becomes longer than the other. The longer chain is then taken to be the correct blockchain and the other one is discarded.

Consensus mechanisms work as a means of ensuring trust between mutually distrusting parties, so the rules of conflict resolution within a consensus mechanism have to be transparent [13]. When a node joins the network, they agree to these rules in the consensus mechanism and can independently verify that every block that is added has followed those rules. Each consensus mechanism has different levels of security, transaction throughput, and computational requirements, which makes them suited for different application areas [14]. Some consensus mechanisms place more emphasis on decentralization over transaction rates and others focus more on security which comes at a computational cost so the right consensus mechanism should be used for a particular application area.

1. Proof of Work Consensus Mechanism

In the proof of work (PoW) consensus mechanism, all the nodes have an equal chance of publishing the next block. It is based on trying to solve a computationally-intensive mathematical puzzle, and the first node to solve this puzzle publishes the next block and receives a reward for doing this [15]. This puzzle can only be solved using trial and error, which means that there is no special algorithm that can solve a proof of work puzzle [7]. A solution to the puzzle is however, easy to verify if it is indeed the correct solution, and this allows other nodes to check if the solution is correct without having to do the intensive work themselves. When a user

completes a transaction in a blockchain that uses the PoW consensus mechanism, that transaction is grouped with other recent transactions from other users into a block. Other nodes on the network that want to publish that block are known as miners, and these miners receive a computational puzzle once there is a block that is ready to be published. The miners then compete to try and be the first ones to solve the puzzle. When a miner finds a solution to the puzzle, it broadcasts its solution to the other nodes in the network, who then verify if the solution is correct before the successful miner is allowed to publish the block and get a reward which is usually in the form of cryptocurrency.

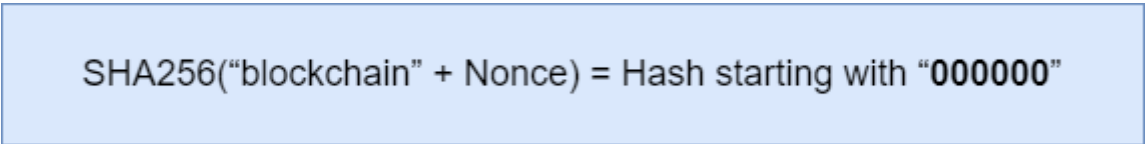

$$\text{SHA256}(\text{"blockchain"} + \text{Nonce}) = \text{Hash starting with "000000"}$$

Figure 2.2: Example of a proof of work puzzle

Figure 2.2 above shows an example of a common PoW puzzle. The puzzle requires miners to find a nonce value that, when appended with a particular given input and run through the SHA-256 encryption algorithm, produces a hash value starting with the specified number of zeroes. The puzzle has multiple correct answers, but the only way to arrive at any of these solutions is to try all possible combinations until the correct one is found. In the example in Figure 2.2, after running the computation, a nonce value of 10730895 produces the correct solution. If the miner starts at 0 and increments by one after each attempt, then this puzzle requires 10,730,896 attempts to arrive at the first correct answer. The difficulty of the puzzle can be adjusted by changing the input string and by increasing or decreasing the number of leading zeros required and Bitcoin, for example, changes the difficulty after every 2016 blocks to keep the rate of new blocks constant at one block every ten minutes. It is easy to check if a possible solution is correct by just running it through the SHA-256 algorithm and checking the number of leading zeros in the hash output [15]. This means that in order to verify a solution, other nodes only have to do a single computation.

The strength of the PoW consensus mechanism is in its decentralization and the security that comes with it. If there are conflicting versions of the blockchain, the longest chain is taken to be the correct one as it has had the most computational work put into it. Suppose a blockchain network using this consensus mechanism has at least 51% of the computational power being controlled by honest nodes, in that case, the correct version of the blockchain will add more blocks to it faster than other versions of the blockchain, which will lead to those versions being discarded. For a malicious user to be able to change a particular block, they would have to redo the computation of that block as well as all the blocks after it to catch up with the correct version of the blockchain and then produce new blocks faster than the honest nodes in order for their version to be taken as the correct one. This is highly unlikely as the estimated total hash rate for Bitcoin is over 150 million Tera hashes per second, and a user would need to have computing power that is greater than half this value to have a chance of manipulating blocks. For context, an Intel Core i5-9600 CPU with a speed of 3.7GHz only has a hash rate of 2380 hashes per second.

The major drawback of the PoW consensus mechanism is that it is resource-intensive. For each potential block that needs to be mined, thousands of processors are all competing to solve the puzzle and get the reward. Since only one node can publish a block, this means that all the other nodes would have used their resources in vain. This means that for PoW-based blockchains with a lot of nodes such as Bitcoin and Ethereum, a lot of electrical energy is consumed for each block that needs to be mined as miners resort to more powerful processors. The annual energy consumption of Bitcoin is estimated to be anywhere between 60 and 125 TWh, and as a comparison, countries like Austria and Norway consume 75 TWh and 125 TWh, respectively, over the same period [16].

2. Proof of Stake Consensus Mechanism

The proof of stake (PoS) consensus mechanism aims to reduce the energy consumption of blockchains by removing the need to have multiple processors all working

at the same time trying to solve a puzzle. It is based on the principle that the more stake a user has in a blockchain, the more they will want it to succeed and the less likely they will act maliciously [7]. PoS then rewards users for having a higher stake in the blockchain by giving them a higher probability of publishing new blocks than users with a lower stake. The definition of stake varies from blockchain to blockchain, but the common methods involve a user investing a certain amount of their cryptocurrency into the system where it is locked, and the user will not be able to use it. They can withdraw it anytime, but then they lose their stake rating in the blockchain.

PoS-based blockchains use different methods to determine the node that will publish the next block, but they all follow the same principle that a user with a higher stake is more likely to publish a block than nodes with a lower stake. The methods include a random selection from the staked nodes and coin aging. In random selection, the node to publish the next block is chosen at random, but the percentage stake that a node has out of the total staked cryptocurrency is the likelihood that they will be selected. So if a node holds a 15% share of the total staked amount, then they have a 15% chance of being selected. With coin aging, the staked amount has an age property attached to it and a rule specifying that the staked amount has to be locked for a specified period before it can be considered to be part of the stake [17]. An example is a 30-day rule which requires an amount to be locked for 30 days before it can be a part of the stake, and this age is reset after a specified period which means the age goes back to zero and the amount has to be locked for another 30 days before it can be part of the stake again.

Since there is no need for powerful processors and the associated electricity consumption, some blockchains that use the PoS consensus mechanism reward the nodes that publish blocks through transaction fees instead of mining rewards [10]. In such blockchains, all the cryptocurrency is already distributed to the nodes as opposed to creating new cryptocurrency through mining which is the case with the PoW. If a node acts maliciously in a PoS blockchain, they lose their staked amount,

and this acts as an incentive for nodes to act in an honest manner. The major flaw of the PoS is that those with a higher stake will most likely get selected more often than those with a lower stake, making them increase their stake and make it difficult for other nodes to catch up.

3. Proof of Authority Consensus Mechanism

In the proof of authority (PoA) consensus mechanism, users' real-world identities are leveraged as a means of ensuring honest behaviour in the network. This means that nodes intending to publish blocks on the blockchain must have their identities verified first. Once a node's identity has been verified, then the other publishing nodes that are known as authority nodes or sealer nodes will vote on whether or not the node should become a sealer node. If more than 50% of the sealer nodes approve, then the node will become a sealer node and be able to publish new blocks. A node can also be removed from being a sealer node if more than 50% of the nodes on the network vote for its removal. The right to become a sealer node should not be easily obtainable, so that sealer nodes value the privilege to publish blocks.

To select one sealer node from the pool of sealer nodes that will publish the next block, a round-robin method is used to ensure fairness, and if a sealer node is not online, then the next sealer node in line is selected to publish the block. If a blockchain has more than one sealer node, then no sealer node may publish consecutive blocks. The time interval between successive blocks is fixed in PoA blockchains, and sealer nodes do not require users to be present to do the validation of each block. The process is automated, and the users just have to monitor that the nodes remain connected to the network so that they can continue publishing blocks. One of the major benefits of PoA-based blockchain systems is that they have a high transaction throughput as there is no mining required. This also makes it very energy-efficient and not computationally intensive, as the process of validating blocks requires very little energy. PoA-based blockchains can even be run on low-specification computers, making them cost-effective as well.

Blockchains that use the PoA consensus mechanism are not as decentralized as

those that use PoW as the power to publish new blocks is bestowed upon a few select nodes. This makes it unsuitable for use in systems that require complete user anonymity, such as permissionless cryptocurrencies, but this feature makes it a good consensus mechanism to use in enterprise applications where there is a high degree of trust. Decentralization within the network can be increased by having a large number of sealer nodes. For a node to be able to act maliciously, they have to control at least 51% of the sealer nodes in the network. This is unlikely since all the sealer nodes in the network are known, and a user can be immediately identified if they try to influence other sealer nodes to act maliciously. Linking a node to a real world individual means that legal remedies can be sought if a user is found to have acted maliciously, which adds an extra security layer.

2.1.6 Types of Blockchain Networks

Blockchains are usually classified into two categories, namely public and private networks, based on who is able to publish new blocks in the network [18] [19]. If a few select nodes can publish new blocks, then it is a private blockchain, but if any node in the network can publish blocks, then the blockchain is public.

1. Public Blockchains

Public blockchains are also known as permissionless blockchains because they do not place any restrictions on the nodes on the network. This means that anyone can join the blockchain without needing any authorization from other nodes, and once they join, they have the same privileges as the other nodes on the network [20]. Every node on the network has both read and write access to the blockchain which enables them to initiate transactions as well as to see other transactions that other nodes have done. Read access is not just restricted to nodes in the blockchain. Anyone can download a copy of the blockchain and view all the transactions that have ever been done on the blockchain, even if they are not a participant on the network. The nodes in a public blockchain can all publish new blocks and validate the blocks that have been published by other nodes [21]. User anonymity is a

big feature of public blockchains as there is no user identification necessary for one to join the blockchain. Users on public blockchains are only identified by their addresses, and these addresses cannot be directly traced back to a real-world entity. This user anonymity is balanced out by complete transaction transparency and the use of open-source software in order to gain the trust of users. Public blockchains use consensus mechanisms that can facilitate transactions between distrusting and anonymous parties, such as the proof of work and, to a lesser extent, the proof of stake. This makes most public blockchains such as Bitcoin and Ethereum resource-intensive and suffer from scalability issues.

2. Private Blockchains

Private blockchains which are also known as permissioned blockchains, are blockchains that restrict who can publish new blocks. Only nodes that have been authorized to publish new blocks can do so, and this authorization may come from a centralized authority or from a group of decentralized group of authority nodes. This authorization can be revoked if a node breaks the rules of the blockchain. In private blockchains, it is possible to have different access levels. A user has to be identified first before joining the network or being approved to publish new blocks, which eliminates the anonymity found in public blockchains. Having a group of known individuals to publish new blocks enables permissioned blockchains to use faster and less computationally intensive consensus mechanisms like the proof of authority. This makes private blockchains an ideal solution for application areas that require a high rate of transactions. Since control of the network lies with a few select nodes, it is possible for them to restrict read or write permissions from other nodes that are not authority nodes.

An approach that classifies private networks into two types based on who can read or write data to the blockchain has been proposed [22]. This approach contends that in private networks, users that are not authorized to publish new blocks may have further restrictions placed on them, such as the ability to initiate transactions or view past transactions that are on the blockchain. The two types of private blockchains

are private and open blockchains as well as private and closed blockchains.

Private and Open Blockchains

In this type of permissioned blockchain, write restrictions are placed on the data in the blockchain. Only nodes that have been authorized to write data to the blockchain can do so, but any node can read the data. A private and open blockchain is still permissioned because nodes need to be identified first before they can join the blockchain, although even after joining the blockchain, they have read-only privileges. They can only view the data in the blockchain but cannot add any data to it. Nodes that have been granted permission to write data to the blockchain may or may not be the same as authority nodes that are responsible for publishing blocks, but every node that can publish blocks also has both read and write privileges. Private and open blockchains can be used in a number of application areas where only a few nodes can write data to the blockchain, but every node can read the data, such as a system for public companies to release their financial statements. Only the nodes belonging to the companies are allowed to post data to the system, but anyone can view the data once it has been written to the blockchain. This type of blockchain can also be used in supply chain management as it places restrictions on who can update the information in the blockchain.

Private and Closed Blockchains

Closed blockchains are the most common type of private blockchains. In these blockchains, only a few authorised nodes can write data to the blockchain, and only authorised nodes can view data written to the blockchain. This type of blockchain is ideal for applications where users want to keep their transactions private from everyone, including other nodes on the blockchain. This can be achieved by creating a private channel between two nodes in a private network where they can transact without exposing details of the transactions to other nodes. There are various ways that the blockchain can be updated after these transactions have been completed. Some blockchain protocols only write a transaction marker to the blockchain that only states that a transaction has taken place without including the details of the

transaction, and other blockchain protocols use a timer for the channels and update the blockchain at the end of the timer with the state of the channel. Some blockchain protocols make it possible to have more than two nodes in a channel, and every node in the channel can transact with other nodes and view all the transactions in the channel. A potential application area for this type of blockchain is for a system of multiple suppliers and buyers where the details of transactions between two parties have to remain confidential.

2.1.7 Blockchain Architecture

This section looks at the various architectures of blockchain that have been proposed by different scholars. The architectures detail the layers that make up a blockchain system.

1. Seven-Layer Blockchain Architecture

The first architecture proposed has seven layers with different functions in the overall blockchain system [23]. The first layer, as shown in Figure 2.3, is the data layer, which acts as the data structure of the blockchain and the data storage area. It is where the blocks of data and encryption algorithms are physically stored. The network layer contains rules about how the nodes broadcast transactions to each other and how the system can make full use of the underlying bandwidth. The physical layer consists of computers, servers, and IoT devices that make up the nodes of the blockchain and are connected in a peer-to-peer architecture. The next layer is the virtualization layer and it is responsible for allocating hardware and other resources to virtual machines. The consensus layer contains the network's consensus mechanism, which are the rules that the nodes use to reach an agreement about transactions in the network. The incentive layer contains rules about how transaction fees or mining rewards are calculated and distributed. The next layer is the services layer which contains optional modules that can be included in the blockchain, such as smart contracts and digital wallets. The API layer acts as an interface between third-party applications and the blockchain, and smart contracts. It provides tools that enable third-party applications to interact with the data in

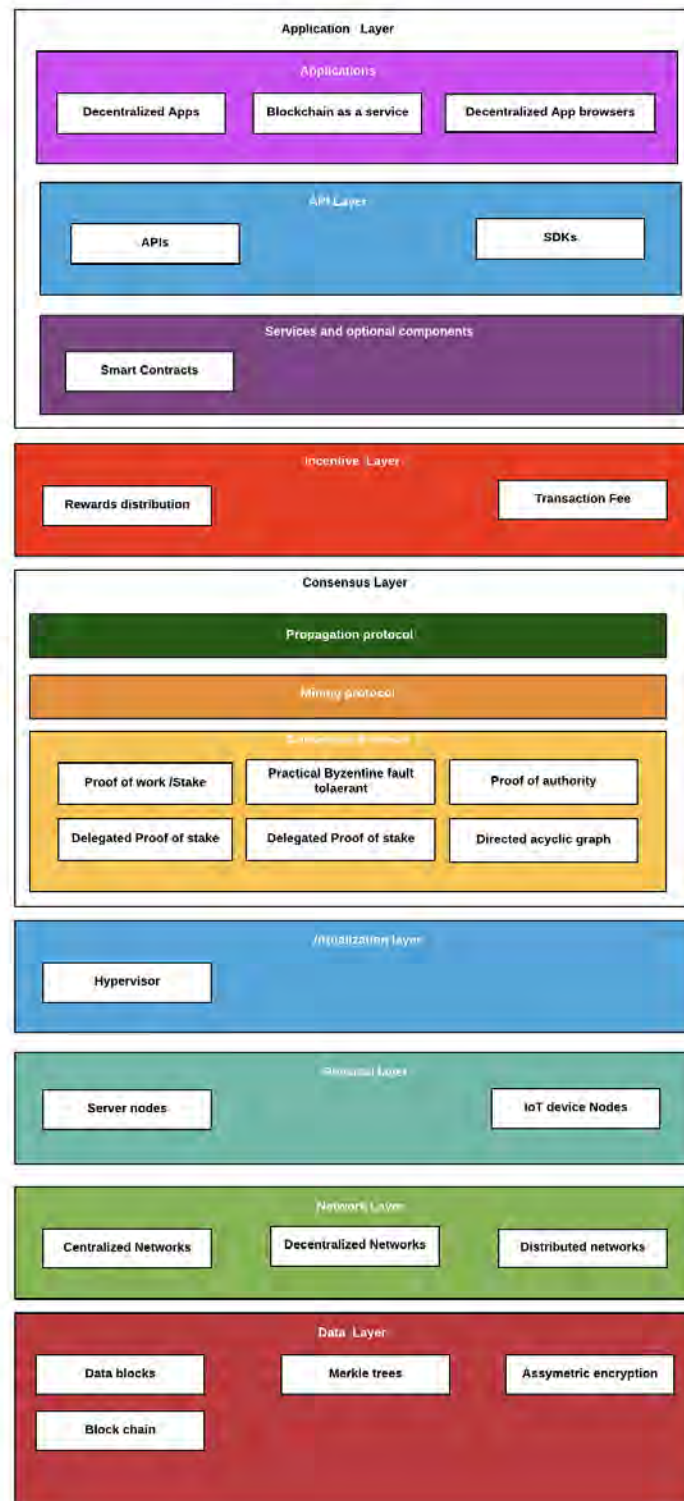


Figure 2.3: Seven layer blockchain architecture [23]

the blockchain. The final layer is the application layer consisting of applications that read data in the blockchain and write data to the blockchain.

A similar 6-layer architecture has also been proposed and used by various scholars [24] [25]. This architecture features the same layers as the 7-layer approach except for the virtualization layer, whose functions are put in the physical layer. They argue that this architecture strikes the right balance between ensuring that every component of the system is accurately represented and its functions are clearly spelled out and making sure that no layer has too many components in it.

2. Three-Layer Blockchain Architecture

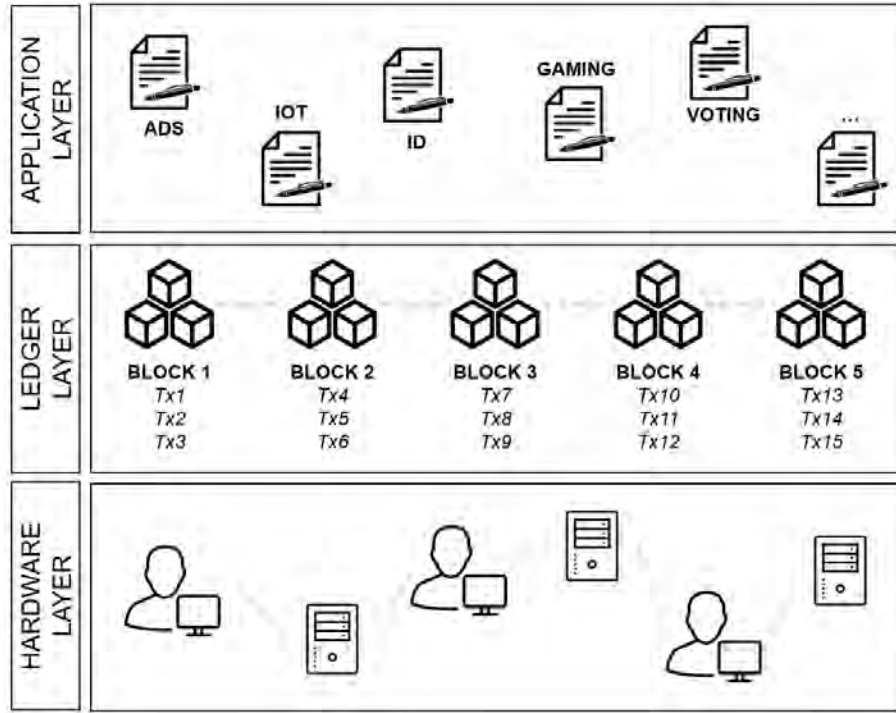


Figure 2.4: Three layer blockchain architecture [26]

A simpler architecture comprising of three layers has been proposed [26]. This architecture consists of a hardware layer, the ledger layer, and an application layer as shown in Figure 2.4. The hardware layer consists of all the nodes in the network. They are responsible for enforcing consensus in the blockchain and to store the digital ledger that records all the transactions. The ledger layer consists of the entire blockchain protocol. This includes records of transactions, consensus mechanism rules as well as all the other rules that govern the blockchain. The application layer

consists of third-party applications that make use of the data in the blockchain.

2.1.8 Blockchain Ethical Concerns

A number of ethical concerns have been raised with regard to the use of blockchain technology. Its decentralized architecture makes it difficult for authorities to monitor the activities on the network. This has led to blockchain being used in the trade of illegal drugs, money laundering, and many other ethically questionable applications [27]. An Ethereum-based application called Augur allows users to place bets on the death of individuals and are paid when this occurs [28], and a website called Silk Road allowed users to buy and sell illegal drugs anonymously, and these transactions were done using Bitcoin [29]. Blockchain-based cryptocurrencies are also the preferred payment method for ransomware attacks. In 2017, more than 200,000 computers worldwide were infected with the WannaCry ransomware, and the attackers only unlocked the computers after they received payment in Bitcoin. Blockchain-based cryptocurrencies such as Bitcoin are chosen for such situations because of the difficulty in tracking the individuals carrying out the transactions as well as the fact that blockchain transactions are irreversible once they have been validated [30]. Another concern from the use of blockchain is its impact on the environment. Blockchains that use the PoW consensus mechanism account for the largest proportion of individuals who use blockchain technology, and these blockchains use large amounts of electricity, which can be detrimental to the environment.

2.1.9 Smart Contracts

A smart contract is a blockchain-based self-executing program that represents an agreement between two or more parties [31]. A smart contract is also defined as a software that is used for authenticating and implementing the terms of an agreement between two parties [32]. It is an if-then system in that the smart contract is executed if and only if the pre-programmed condition is met. Smart contracts allow parties to enter into an agreement without the need to establish trust first or having to rely on a trusted third party [33].

Smart contracts run on a blockchain, and because of that, they inherit some features of

blockchain, such as immutability. Like other transactions on a blockchain, smart contract code cannot be changed once it has been deployed to the blockchain. Suppose a smart contract that has been deployed to a blockchain has an error in its code. In that case, this error cannot be rectified directly on that smart contract because of the immutability characteristic of all the data stored in a blockchain. The only way to rectify such an error is to deploy a separate smart contract with the corrected code and redirect users to it instead [34]. The old smart contract will still be there on the blockchain. Smart contracts are also decentralized because they are stored on the digital ledger, which every node in the blockchain has a copy of as opposed to being stored on a single server like traditional applications. Smart contracts can be autonomous in that there is no need for any interaction between them and the user who deploys them. They can also be programmed to offer services to generate funds as well as spending them when the need arises, which makes them self-sufficient [24].

The use of smart contracts has numerous advantages for users. Smart contracts have a lower risk factor when compared to centralized applications because of the combination of immutability and traceability. The terms of the agreement between the parties in a smart contract cannot be altered, and every transaction is auditable and traceable, which reduces the risk of financial fraud [35]. By removing the need for a trusted middleman and being able to execute automatically, smart contracts have lower administration costs. This can also improve business processes' efficiency as there is no need to keep running everything through specific individuals for them to approve first when the conditions that warrant a particular action have been met [36].

Since smart contracts are dependent on blockchain, they suffer from similar limitations with blockchain. They suffer from limited scalability and performance because smart contract deployments and executions are treated like any other transaction, which means that they have to go through the same validation processes that other transactions are subjected to. Smart contracts also have an issue of irreversible bugs due to the permanent nature of blockchain transactions. Once a smart contract is deployed with bugs, then it is going to stay like that forever. A new corrected version may be deployed and used,

but the other version will always be there on the blockchain as well. Another issue highlighted by Wang et al. [37] is the lack of standards when it comes to smart contract development. A smart contract designed and deployed on one blockchain protocol cannot directly interact with smart contracts that are on a different blockchain protocol. This reduces the potential impact of smart contracts for everyday use as it would require the majority of users to be using the same blockchain protocol.

1. Smart Contract Use-Case

This section describes the steps and the parties involved in the development, deployment as well as execution of smart contracts.

The Participants are as follows:

- **Developer** – This is the individual who writes the smart contract code and deploys the smart contract to the blockchain.
- **User** – The user is a participant in the blockchain who calls the smart contract once it has been deployed to the blockchain.
- **Nodes** – The nodes represent the other users in the blockchain. The nodes are responsible for the validation of transactions in the blockchain.

Figure 2.5 illustrates how a transaction that involves interacting with a smart contract is handled. When a user wants to call a method contained in a smart contract, it is treated like any other transaction meaning the user needs to know the address of the smart contract just like they would need to know the address of any other account that they intend to transact with. The user broadcasts the transaction details to all the other nodes in the blockchain and this transaction is then included in a candidate block that needs to be mined. Depending on the consensus mechanism used by the blockchain, other nodes will compete to mine the block. When one of the miners has successfully mined the block, the smart contract is executed, and the user receives confirmation that their transaction has been completed and a record of that transaction will now be on the blockchain.

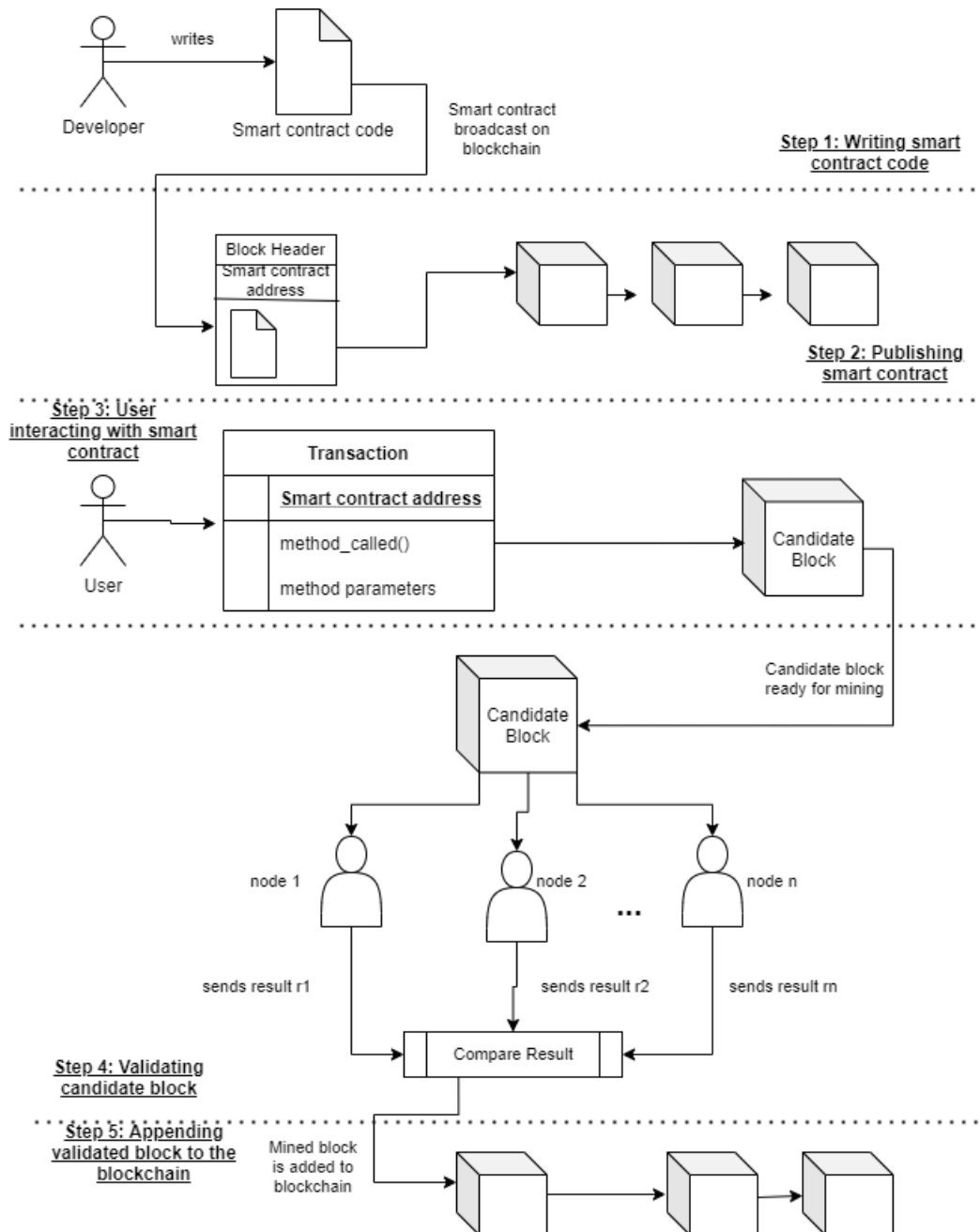


Figure 2.5: Smart contract use-case diagram

2.1.10 Blockchain Summary

Blockchain technology has been described in great detail and the different technical aspects have been compared. The decentralized architecture of blockchain makes it an ideal technology to use for a peer-to-peer system such as the one proposed by this research. Different consensus mechanisms were compared and the effect they have on the overall performance of the system in which they are used was also discussed. This understanding of their different power and processing requirements as well as the security levels and transaction throughput aids in making more informed design choices that are more suitable for a particular application area.

2.2 Internet of Things Overview

The internet of things (IoT) can be defined as the ability of any device with networking capabilities to sense and collect data from different areas in the world, and then upload that data to the internet where it can be processed and stored [38]. The internet of things can also be defined as a concept of connecting any electronic device to the internet. This ranges from devices that have traditionally been able to connect to the internet like GPS devices and digital cameras to devices that did not previously have that feature like washing machines, coffee makers and refrigerators among others [39]. The internet of things is not just limited to devices but also includes components of large systems and machinery like the engine of an aeroplane or the cooling chamber of a nuclear power plant [40].

2.2.1 Internet of Things Architecture

The general IoT architecture consists of three basic layers, namely the perception layer, the network layer and the application layer [41] [42] [43]. The architecture has a bidirectional flow of instructions and data as the data moves from the perception layer up to the application layer and the instructions move in the opposite direction from the application layer to the perception layer as illustrated in Figure 2.6.

1. Perception Layer

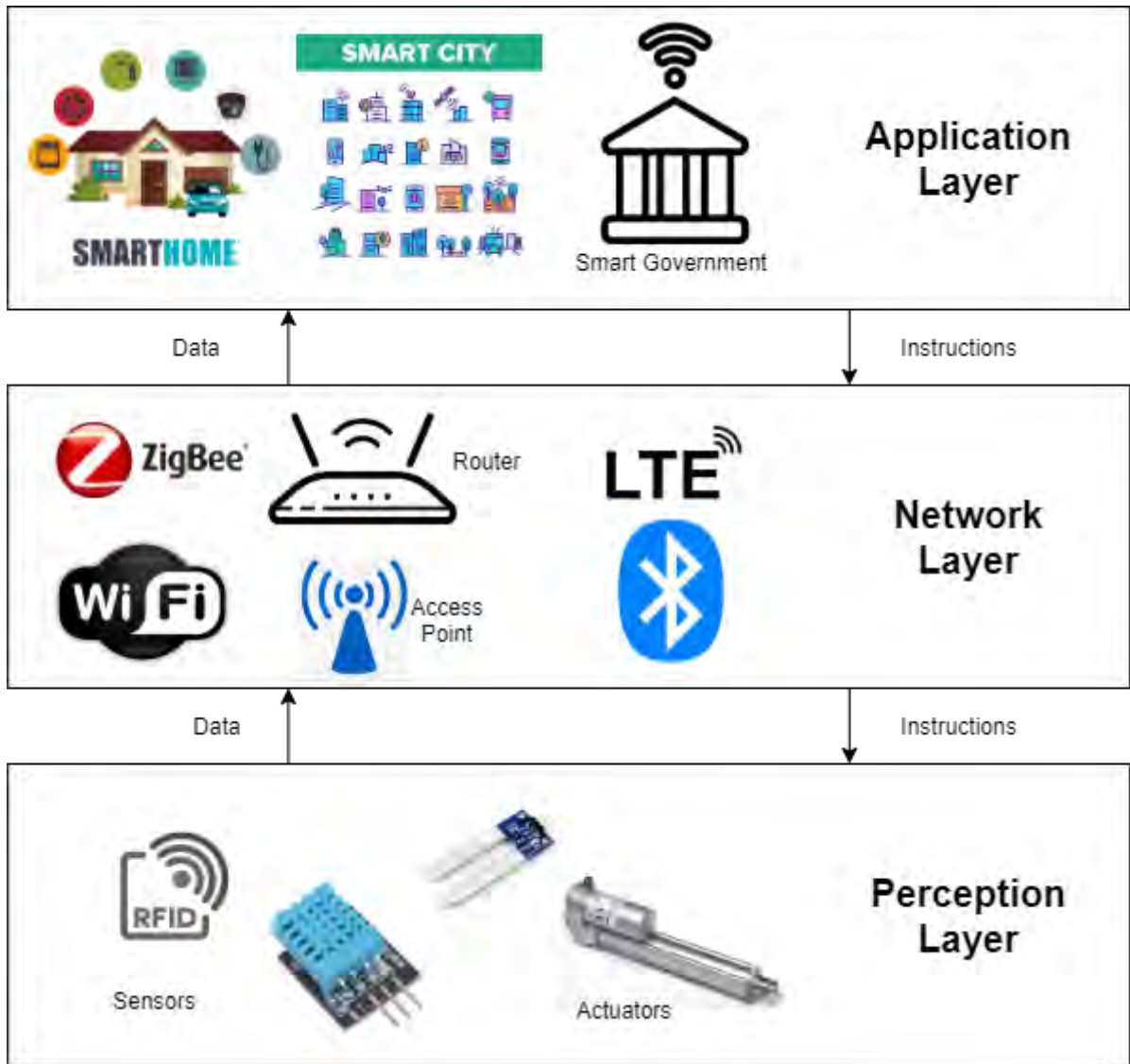


Figure 2.6: IoT Architecture [42]

The perception layer contains the physical devices that are responsible for gathering data about their environment. These devices include sensors such as humidity sensors, RFID sensors and temperature sensors. Devices in this layer usually have limited storage and processing capabilities, and so they have to send the data that they have gathered to other devices like data warehouses or cloud service providers for processing and storage. Another component found at this stage is an actuator that can be defined as a device that changes a particular physical condition like shutting off the power supply or adjusting the temperature in a room [44]. Actuators allow IoT systems not only to gather data about an environment but also to act on that data. An example is a temperature sensor in a room that sends temperature

data to a cloud server for processing, and the server detects that the temperature is too high. The server then sends an instruction to the actuator to adjust the temperature to a certain level.

2. Network Layer

The next layer is the network layer which is also known as the transmission layer. The network layer acts as a gateway for receiving and transmitting the data that has been gathered in the perception layer. It consists of physical devices such as routers and switches that handle the transmission of the data to the correct destination as well as different communication technologies like ZigBee, Bluetooth and Wi-Fi. The various communication technologies found at this layer are meant to cater to the different types of communication technologies that are used by the devices in the perception layer as well as those in the application layer. This makes the network layer the most important layer in the IoT architecture [43].

3. Application Layer

The application layer receives the data that has been collected by the sensors in the perception layer and processes, analyses, and stores the data in databases and data warehouses. The application layer is also responsible for maintaining the confidentiality, availability and integrity of the data [41]. This layer has applications that make use of the data after it has been processed. Examples of such applications include smart homes, smart grids, and smart cities. The success of an IoT system requires these applications to make full use of the data from the sensors.

2.2.2 Benefits of IoT

The use of IoT has brought about an increase in the amount of data that has been collected. Devices that did not previously have connectivity capabilities are now able to capture data and transmit it for processing. This has enabled decision-makers to make better-informed decisions. Another advantage brought about by the use of IoT has been the ability to monitor certain environments remotely. Dangerous tasks such as monitoring the radiation levels in a nuclear power plant can now be done without exposing anyone

to the risk as sensors can be placed and monitored remotely. IoT has also enabled the automation of a lot of processes in a wide range of application areas. IoT systems can be connected to cloud-based applications that will process data received from the sensors and send feedback to some actuators based on the results of the data after it has been processed, and the actuators will act on the feedback without any human intervention. An example of a fully automated system is an irrigation system that has sensors in the soil to check for the amount of moisture. This data is sent to a server for processing, and the server will check the weather forecast, and if there is no rain expected, it will send a command to the actuators to open the sprinklers. All this is done without any human involvement.

2.2.3 Drawbacks of IoT

The major drawback of IoT is the lack of standards of compatibility for the devices. Different devices use different connection technologies such as Bluetooth, ZigBee, Wi-Fi, and 4G as well as different connection ports like the type of USB used. IoT sensors and devices also have different power requirements, such as 3.3V for some devices and 5V for others. IoT systems also collect a lot of potentially sensitive data, and so privacy is a concern in such systems. If an unscrupulous individual gains access to this data, it can lead to more serious issues like identity theft. IoT systems are complex to design as the design process involves trying to integrate many different components, which poses a risk of a faulty system design. Since IoT systems are used in critical application areas such as water treatment plants, this leaves very little room for errors.

2.2.4 IoT Summary

The architecture of IoT was discussed as it forms one of the base architectures of the proposed system. The proposed system has different components and each of these components can be directly attributed a layer on the IoT architecture. This is to demonstrate that the proposed system builds on previous researches that have investigated IoT technology to come up with the three-layer architecture.

2.3 Related Work

This section looks at how blockchain and IoT technologies have been integrated into different application areas, including the energy sector. It also evaluates other similar studies that have been carried out, critically assess them and justify why this study is necessary.

2.3.1 Blockchain Use in IoT-Based Systems

Traditional IoT implementations are centralized in that they require servers for processing and data storage. The drawback of this is that the servers become a bottleneck that can cripple the whole network if they go offline [45]. Decentralizing IoT networks through the use of a blockchain peer-to-peer model helps mitigate the issue of a single point of failure [46]. The strong cryptographic features of blockchain have also made it an ideal means of providing security and maintaining data integrity in IoT systems [47]. There are, however, a number of challenges that may inhibit the successful use of blockchain technology in IoT systems.

A study to identify some of these challenges and how they can be mitigated against was carried out by Zorzo et al. [48]. The first issue they identified was the hardware limitations of IoT devices. To store a copy of a Blockchain requires a large amount of storage as most of them are very large, with Bitcoin being more than 200 gigabytes [49] [50]. A blockchain becomes larger over time as it stores a record of every transaction that was ever conducted on the network. Another limitation of IoT devices that was identified is their limited computational power [51]. Blockchain networks that use the Proof-of-Work consensus mechanism require a lot of processing power as the consensus mechanism involves brute force calculation which would take years to solve using current IoT devices. To find a way around these issues, various authors proposed a number of solutions. The first possible solution is to create a hierarchical peer-to-peer network where there are two different classes of nodes [48]. The first class has nodes with full computing power, and these are responsible for controlling the IoT devices as well as communicating with other full nodes for the maintenance of the blockchain network. Another proposed

solution involves using a blockchain that is completely separated from the IoT devices. This means that the blockchain is not hosted on the IoT network, but each of the IoT devices has access to it and is able to make transactions. The final proposed solution called for the use of smart contracts on the nodes that have full computational power. The IoT devices only send raw data to these nodes that will then execute the smart contracts based on the data. This reduces the processing requirements of IoT devices while ensuring that it is integrated with a Blockchain network.

A few studies have been carried out that have managed to mitigate against these challenges and successfully integrated blockchain into IoT systems. Sun et al. have come up with a model to successfully integrate blockchain onto a wireless IoT system. The model breaks down nodes into two types of nodes [52]. It has transaction nodes consisting of low-power IoT devices and fully functional nodes with high storage and processing power. Transaction nodes can initiate a blockchain transaction which will then be processed by the full nodes before the transactions are added to the blockchain. To keep the system decentralized, a transaction has to be broadcast to all the full nodes, which should all have a copy of the blockchain ledger. The model was tested against common network attacks such as an eclipse attack and random link attacks, and it was found that the structure of the network allowed it to withstand the attacks against it.

A system that uses blockchain and IoT technology to check for data integrity was designed and implemented by [53]. The objective of this system was to provide device owners with an immutable solution to check if their data has been accessed or amended. The system uses Raspberry Pi devices as the servers for the IoT devices and smart contracts built onto the Hyperledger Fabric network running on desktop computers. Hyperledger Fabric is a permissioned decentralized blockchain framework that allows developers to develop smart contract-based systems [54]. This study argued that IoT required a high throughput blockchain network and so proof of work-based blockchains would not be ideal for IoT applications. They advocated for the use of voting-based consensus mechanisms like the practical byzantine fault tolerance (PBFT) mechanism or the crash fault tolerance (CFT) mechanism. The study highlighted the possibility of using blockchain

for applications with a high rate of transactions and how the choice of the consensus mechanism influences this. The main disadvantage of this study, as with many systems that attempt to integrate IoT with blockchain, is that not all the devices in the system are on the blockchain network. The Raspberry Pi devices are limited to just processing IoT data and then sending the data to other nodes that are in the blockchain that will then add the data to the blockchain.

Blockchain has been put forward as a solution for the security issues that come with the use of IoT [55]. The study argues that the decentralized architecture of blockchain would make it highly unlikely to attack an IoT system built on a blockchain using a denial of service (DOS) attack. The paper also suggests that this characteristic of blockchain would eliminate any system downtime. The various consensus mechanisms such as the PoW and the PoS make it almost impossible to create fake nodes and initiate a Sybil attack. This paper highlights that not only is it possible to use blockchain technology with IoT but that it is highly beneficial to do so from a security standpoint.

Dai et al. carried out a comprehensive survey to see if there are any benefits to using blockchain in pre-existing IoT systems [56]. They identified some advantages to using the two technologies together, such as enhancing the interoperability of IoT. This is enabled by the fact that blockchain networks are built on peer-to-peer overlay networks that support widespread access to the internet. Another benefit of using IoT in conjunction with blockchain is the traceability and reliability of data from IoT devices. Data in a blockchain is immutable and traceable in that once data is stored in the blockchain, it cannot be changed or deleted without invalidating the entire chain of data, and the transactions on the blockchain are transparent. Blockchain can also enable autonomous transactions in IoT systems through the use of smart contracts. For example, two companies can trade with each other and have payments executed automatically using smart contracts without human intervention, which can save the companies a lot of money.

2.3.2 Blockchain Use in the Energy Sector

Blockchain networks are known to use huge amounts of energy [57] [58] [59], and so their application in the energy sector has been limited because of this. A single Bitcoin transaction uses enough energy to power eight households for a full day [60]. This presents a challenge in trying to incorporate such technology into systems designed to conserve energy. A number of solutions to this challenge have been presented.

The first method is to use a consensus mechanism that is not as energy-intensive as the proof of work mechanism [61]. The PoW algorithm rewards users for being the first to solve a complex mathematical puzzle [62]. This creates a scenario where hundreds or thousands of processors are working at the same time, all trying to solve the puzzle. This puts a huge strain on the electricity grid as these powerful processors have big energy demands [63]. The PoA consensus mechanism is proposed as a viable alternative to the PoW mechanism [64]. In the PoA, a low number of nodes are selected to validate the transactions in the blockchain. This ensures that users do not have to compete for anything, thereby significantly lowering the network's overall energy consumption.

The energy consumption of blockchain can also be lowered by the consolidation of transactions between parties which will then be recorded as a single block. This is done through the use of state channels which can be defined as processes that allow users to transact directly with each other outside the Blockchain [65]. A state channel can be used, for example, in a Blockchain-based voting system to store the individual votes cast and only recording them on the blockchain once all the votes have been cast. This reduces the number of transactions on the actual blockchain, which in turn reduces the energy consumption of the system as a whole.

2.3.3 Evaluation of Related Energy Studies and Projects

This section evaluates studies and projects that have used IoT and blockchain in the energy sector and identifies the successes and shortfalls of each study and how this research can build on those studies.

Jain and Dogra came up with a system to distribute energy harnessed from solar

systems in a decentralized manner [66]. The system uses IoT-enabled devices to get information from batteries about how much power is stored in them, and this information is uploaded to cloud databases for storage. Blockchain is used to keep the system decentralized as users can transact with each other without the need for an intermediary. To enable the users to be able to trade energy with each other, the authors created their own cryptocurrency known as Zoncoin. Households that produce energy are paid using this cryptocurrency which is also used to reward nodes that process transactions. The system uses the PoW consensus mechanism in order to increase the time it takes to create new blocks and reduce the chances of malicious blocks being added. An Arduino microcontroller is used to process the data from the IoT devices and update the data in the cloud database. Users who need to buy energy use an application to check for other users that have excess power, and they can purchase the energy on this application. After the transaction is complete, it is added onto the blockchain, and the IoT device receives a command to transfer the energy between the two parties. This study has a few drawbacks. The first issue is the use of the PoW consensus mechanism and placing restrictions on the frequency of block creation. The PoW consensus mechanism consumes a lot of electricity, and the mining processes might end up consuming the majority of the energy that is produced by the entire system. The second issue is the use of the system's own currency in the form of the Zoncoin cryptocurrency. The cryptocurrency will have to have a similar value with the energy produced and be easily tradeable for a currency that is usable elsewhere, otherwise the participants might feel like they are getting a valueless token for their energy. This system, however demonstrates the potential there is in the development energy trading systems. The system proposed by this research will use smart contracts to facilitate payments in the trading of energy in a similar way that was demonstrated in this study.

Wang et al. designed a model for a distributed energy system that uses a peer-to-peer network to exchange information and energy [67]. The system is used to manage the demand for energy by residential and industrial users by making use of smart contracts. The system is designed to automatically transfer excess energy from one consumer to

another consumer that has a higher demand at that particular time. This system is different from other energy trading systems in that here, the energy is not sold to the utility company but is transferred directly to other consumers without the seller and the buyer having to approve the transaction first. Instead, the system automatically facilitates the transactions based on the supply and demand of energy using smart contracts. One of the challenges of the large-scale implementation of this system that was highlighted is the inability of the Blockchain network to support high-frequency energy trading transactions. This system does not give any options to the buyer to set their own price or when to sell, and the buyer cannot choose the seller they want to buy from.

A system that allows users to sell off excess energy was developed by Hwang et al. [68]. The system analyses the energy usage patterns of consumers in order to improve efficiency. IoT devices were used to collect data on how much energy a consumer is using at a particular time, and this data was sent to cloud storage facilities where it can be analysed to come up with usage patterns. The model used Blockchain technology for transactions to enable consumers that generate surplus energy from renewable energy sources to be able to sell it to the utility company. This study provides great insight into measuring and analysing the amount of energy that consumers use and using this to determine how much excess energy they have. The disadvantage of this system is it has a single entity that purchases all the excess energy in the form of the utility company. This gives the utility company a monopoly over pricing for the energy, and everyone will have to buy from the utility company.

Another study that used IoT and blockchain to measure energy consumption accurately was carried out by Gao et al. [69]. They leveraged blockchain's ability to provide a medium of accountability between two entities that do not trust each other and used it to provide more transparency between electricity providers and consumers. The system they developed allows users to monitor their electricity usage accurately without any fear of the data being manipulated by anyone, including the electricity provider. A smart meter is used to record the data about the power usage, and this data is encrypted and then sent to the electricity provider, where a smart contract decrypts it before it executes

a procedure based on the data. If a consumer runs out of electricity credit, the smart contract executes a procedure to shut down the electricity and then sends a message to the electricity company, and this transaction is stored on the blockchain.

Li et al. and Ahl et al. carried out studies that broke down the main power grid into smaller grids known as micro-grids that have the ability to trade power with each other [70] [71]. Both these models eliminate the middleman and use smart contracts to reward micro-grids that manage the network by checking that all the other grids are up and that there are no unauthorized users. The rewards are in the form of electric power, and micro-grids also get rewarded for maintaining the efficiency of the entire network. Each micro-grid has to be kept sufficiently small to address some of the scalability issues that are often associated with blockchain systems. Both studies provide good frameworks for blockchain-based peer-to-peer energy trading, and this research will build on the results from both studies that demonstrate a fully decentralized system with minimal interference from a central authority. This principle will be used in the proposed system, albeit among peers in a single micro-grid.

2.3.4 Conclusion

This chapter has evaluated the architectures of both IoT and blockchain and showed how they are being used, both separately and together in various sectors. The chapter also assessed some related works and identified ideas from those studies that this research can build on.

Chapter 3

System Architecture

This chapter looks at the approach that was taken in the designing of the proposed peer-to-peer energy trading platform. The architectures of the different parts of the system are also be evaluated, and some design choices are made on the appropriate technology to use for the various parts of the system.

3.1 Design Approach

Traditionally, there have been two broad approaches to system design, namely top-down and bottom-up design [72]. Each of these has its own advantages and suitable application areas, and they are evaluated below to select an appropriate approach for the design of the peer-to-peer energy trading system.

3.1.1 Top-Down Design Approach

In the top-down design model, the overview of the system is articulated, and the sub-systems are specified without being too detailed. All the sub-systems are then broken down, and their sub-systems are identified and specified. This process is repeated until the system has been broken down into its base elements [73]. Once the system is broken down into its base elements, then the modules can be built starting with these elements. The base elements are then put together, and the rest of the system is built from these modules. This approach requires a complete understanding of the whole system, and planning is emphasized [74]. Under a top-down approach, system development cannot start before the system or some part of the system has been broken down to its base

elements.

3.1.2 Bottom-Up Design Approach

The bottom-up design model uses a modular approach by designing the lowest sub-systems first and combining these sub-systems to develop more complex systems. These systems are then integrated, and they become sub-systems of the next level system. This process is repeated until all the components are combined into a single system [75]. In this approach, it is possible to reduce redundancy and hide low-level details of the implementation. Another advantage of the bottom-up approach is that the low-level modules that are developed first are reusable in other parts of the system. The bottom-up approach is the ideal model for the peer-to-peer energy trading system as the system has clearly distinct modules that can be designed separately and then combined to make the complete system.

3.2 High-level System Design

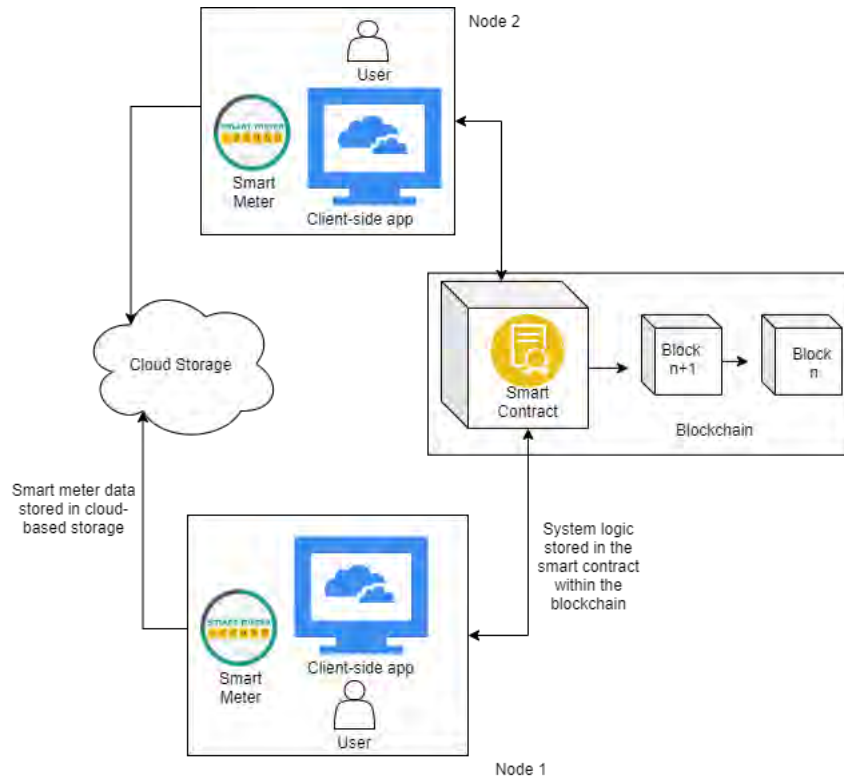


Figure 3.1: High-level system design of the peer-to-peer energy trading system

The proposed system, as illustrated in Figure 3.1, is a blockchain-based peer-to-peer energy trading platform that allows users to buy and sell excess energy to other consumers without a third party. A smart contract is used as the system's main logic, and it handles all the transactions between the parties. The smart contract is also used to store the list of all the energy listings from sellers, and the smart contract is deployed on a permissioned blockchain in order to limit access to only authorized users. The blockchain also facilitates payment between parties through the use of cryptocurrency. The system uses a cloud-based storage facility to store data from a smart meter that records a user's energy consumption as well as how much energy the user has left. Both buyers and sellers interact with the system through a client-side application that is also hosted on the cloud. The client-side application gets data from both the smart contract as well as the cloud storage and displays it to the user. The entire system's architecture is decentralized, and there is no single entity that has control of the system. All the members of the system are active participants that can both buy and sell energy. The management of the system is the participants' responsibility, but the system is designed to reduce the need for regular maintenance.

3.3 Peer-to-peer Energy Trading System Use-Case

The purpose of the use-case diagram is to identify all the active participants in the system as well as their roles in the system. The use-case diagram also identifies some of the processes in the peer-to-peer energy trading system. Figure 3.2 shows the use-case diagram for the proposed peer-to-peer energy trading system.

The participants are as follows:

- **Seller** – The seller is the one who lists their excess energy for sale on the web application.
- **Buyer** – The buyer is the entity that intends to purchase energy from other sellers.
- **Authority node** – The authority node is the node on the blockchain that is responsible for validating transactions before they are added to the blockchain.

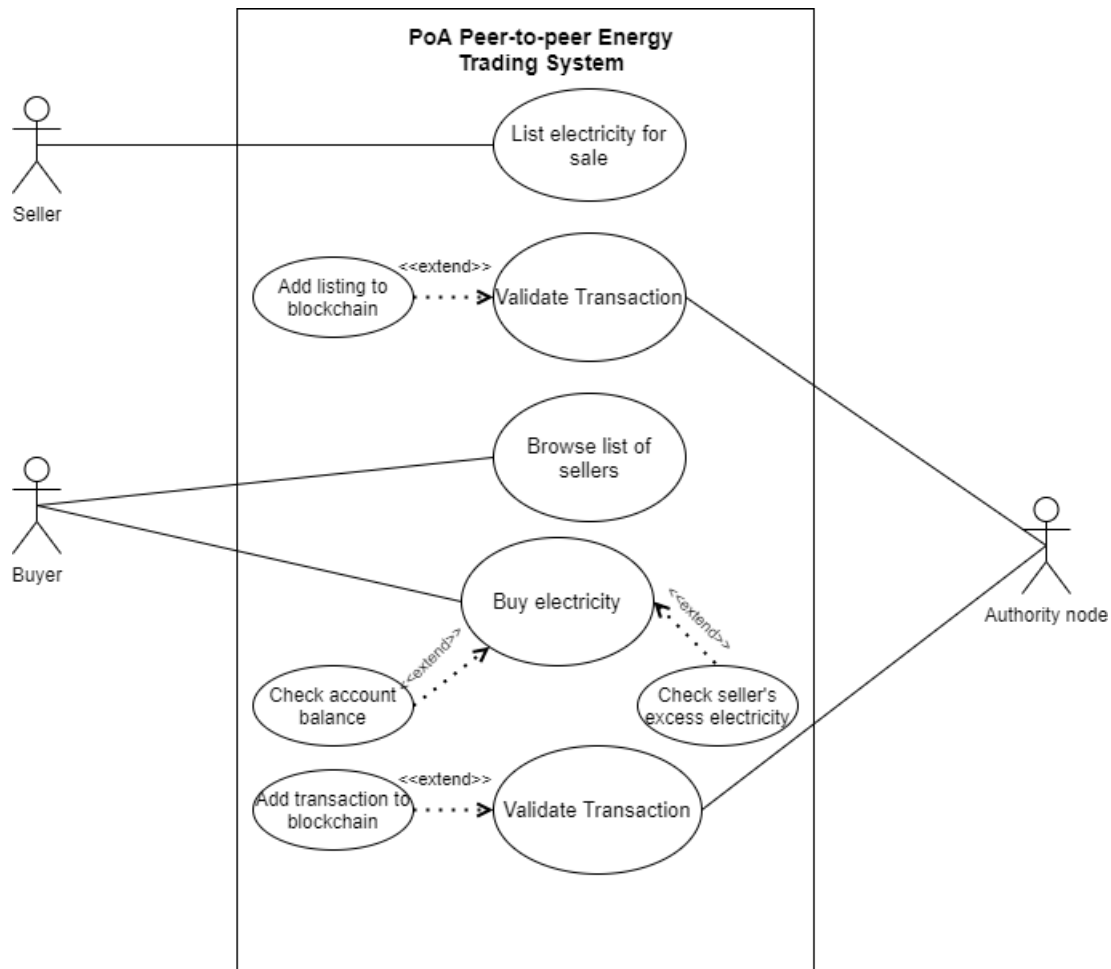


Figure 3.2: Use-case diagram of peer-to-peer energy trading system

The seller accesses the web application and adds a listing. This listing includes the amount of energy they want to sell and the price they want for it. Since this has to be recorded on the blockchain, it is considered a transaction, which means it must be validated before it can be added to the blockchain. The authority node is responsible for validating transactions in a PoA-based blockchain. Once this validation is complete, the seller's listing will now be visible on the web-based application along with other listings. When a potential buyer accesses the application, they can see these listings that contain the seller's address, the amount of energy being sold, and the price. When the buyer chooses a particular listing, the system will query the seller's smart meter and notify the buyer if the seller has the amount of energy that they intend to sell. The system will also check if the buyer has enough money in their account to make the purchase. If both checks are affirmative, then the purchase will go ahead. Once the transfer of energy

is complete, the smart contract will transfer the money from the buyer's account to the seller's account. The authority node will again validate this transaction before it is added to the blockchain.

3.4 Peer to Peer Architecture

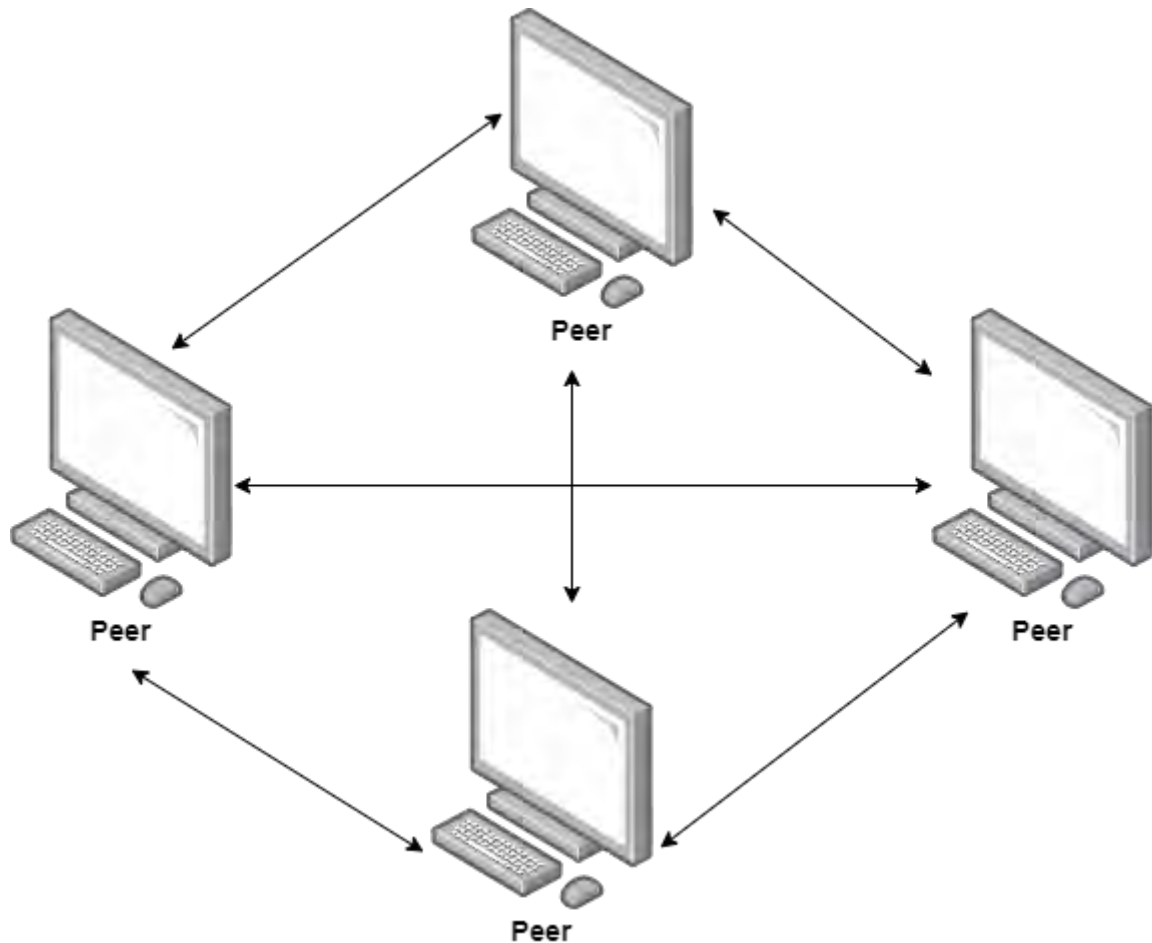


Figure 3.3: Illustration of peer to peer architecture

The nodes in the proposed system are connected to each other in a peer-to-peer architecture. Figure 3.3 shows an illustration of four nodes connected to each other using a peer-to-peer architecture. A peer-to-peer architecture is a decentralized network where the nodes are connected to every other node on the network, and they share the resources and workload amongst each other [76]. This contrasts with the client-server architecture, where the server handles requests and distributes resources to the client nodes [77]. In a peer-to-peer architecture, all the nodes are both clients and servers, meaning they can

request as well as provide resources to other nodes. This architecture is more resilient to complete failure as nodes can join and exit without altering the efficiency of the network as opposed to the single point of failure in the client-server architecture. Nodes in a peer-to-peer architecture are not necessarily equal as some nodes may have extra functions such as the maintenance of the network, and some nodes may contribute more resources to the network, but this does not give them overall control of the network in the same way that a server has control of the network in a client-server architecture. This architecture is ideal for the proposed system as it complements the decentralized nature of blockchain.

3.5 Software Architecture

In this section, the various software layers of the peer-to-peer energy trading system are looked at, as well as how they interact with each other. The software tools that are found at each level are also be evaluated and design choices are made on the appropriate tools to use. Figure 3.4 shows the layers that are found in the software architecture of the proposed system.

3.5.1 Blockchain Protocol

The first layer of the system architecture is the blockchain protocol layer. The blockchain protocol layer of the proposed system is found on the data layer of the 7-layer blockchain architecture that was described in section 2.1.7. This layer contains the rules that make up the blockchain data structure and other tools that are used to set up a node and create addresses to store cryptocurrency. The primary function of the blockchain protocol layer in the proposed system is to provide tools for creating a node as well as providing tools that allow a node to set up as well as join other blockchain networks. Another functionality found at this layer is the ability to check the account balance of the addresses in that particular node. There are various blockchain protocols that can meet these requirements, and they are evaluated below before a choice on the most appropriate protocol for the proposed system is made.

1. Hyperledger Fabric

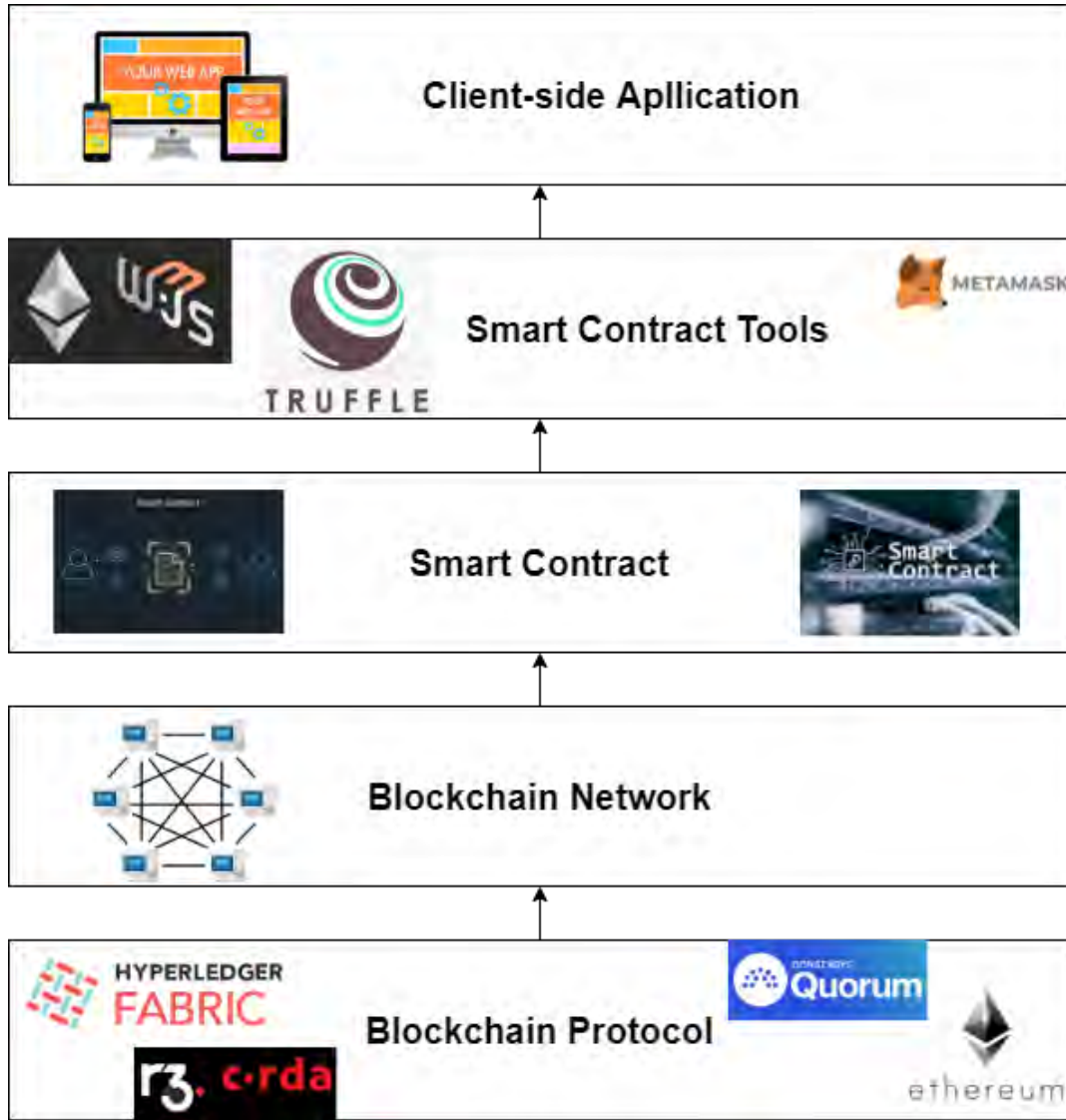


Figure 3.4: Software architecture of the peer-to-peer energy trading system

Hyperledger Fabric is an open-source platform developed by the Linux Foundation for creating and deploying permissioned blockchain networks [78]. Hyperledger Fabric uses the Practical Byzantine Fault Tolerance (PBFT) consensus mechanism, which allows for a high transaction throughput of up to 3000 transactions per second [79]. Since the blockchain networks in Hyperledger Fabric are permissioned, the participating members need to be identified first before they are allowed to transact in the network. This makes it an ideal solution for enterprises that handle sensitive data, such as banks and health institutions. Hyperledger Fabric also has another layer of privacy by having access control within a network through the use of channels. These are communication links between multiple nodes within the

network through which they can transact with each other without other nodes in the network being able to see these transactions [80]. Nodes have to be authorized first before they can join a channel. Hyperledger Fabric provides a platform for the execution of distributed applications on the blockchain network, and these applications can be written in multiple supported programming languages such as Go, JavaScript, and Java, which offers more flexibility than other platforms that usually use a native programming language. It also does not have a native cryptocurrency as there is no mining involved on its networks, making the networks highly scalable.

2. Ethereum

Ethereum is an open-source, Blockchain-based peer-to-peer software platform that allows users to develop and deploy decentralized applications [81]. The Ethereum network is a global network of interconnected nodes. Nodes enforce all the rules of the system through their participation in the validation of transactions, and Ethereum networks support different consensus mechanisms. In Ethereum networks that use certain consensus mechanisms such as the PoW, nodes are rewarded for this validation by receiving the Ether cryptocurrency. In addition to the native Ether cryptocurrency, Ethereum also allows the creation of additional digital tokens through smart contracts [82]. Transactions that can be done by nodes on an Ethereum network include sending and receiving Ether as well as validating other transactions, which is known as mining. Nodes are also able to deploy distributed applications known as smart contracts to the Ethereum network and call smart contracts that have already been deployed to the network [83]. Ethereum has a public network known as the Mainnet, and it also supports the development of smaller private networks as well as public test networks [84]. The different types of networks supported by Ethereum make it a very flexible platform that is suited to a wide variety of application areas.

3. Quorum

Quorum is a permissioned blockchain platform built on Ethereum by JP Morgan. It is a fork of Ethereum which means it is based on the Ethereum codebase but

with certain changes to make it better suited for certain application areas [85]. It shares similar features to Ethereum, such as the use of Ether cryptocurrency and smart contracts. Quorum, however, has some key differences to Ethereum [86]. It limits who can participate in a blockchain network to only those that have been authorized. It further limits privacy by dividing the ledger within a network into public and private ledgers. All the nodes in the network can view the public ledger, but the private ledger is only visible to the parties in a transaction, with only the hash of the transaction appearing on the public ledger. Quorum also uses a voting-based consensus mechanism known as QuorumChain. Consensus is achieved through a smart contract that assigns voting rights to nodes that will then decide on which block gets accepted to the blockchain. The node that created the block does not participate in the voting process, and the smart contract is responsible for tracking the votes. Another difference it has with Ethereum is that in Quorum networks, there are no transaction costs.

4. R3 Corda

R3 Corda, which is popularly known as Corda, is a permissioned and open-source blockchain platform that is primarily intended for use in the financial services sector [87]. It supports the use of smart contracts that link business logic and data with legal principles to make sure that the agreements between the nodes are legally enforceable. Corda gives each network the choice to choose its own consensus mechanism. This is reflected in the ledger by tagging each smart contract with the consensus mechanisms that it uses as it gets added to the network. This makes Corda highly flexible and suited to a wide range of business needs.

Selecting the Ideal Blockchain Protocol

The most important factor when choosing the ideal blockchain protocol for the peer-to-peer energy trading system is that the protocol must support the development and deployment of smart contracts, as this will contain all the logic for the system. All the above-mentioned blockchain protocols support the development of smart contracts. The next factor is the ability to support a high transaction throughput system. The most

important determinant for this is the consensus mechanism used, and the ideal protocols are the ones that support more than one consensus mechanism. R3 Corda and Ethereum both have support for multiple consensus mechanisms. Another factor to consider is the amount of technical support from the developers as well as from the community of users. Ethereum has the most significant support community because it was the first blockchain protocol that had support for the development of smart contracts. The last factor to consider is the ease of use of the protocol in a peer-to-peer system. Most of the protocols listed above are tailored for business-to-business (B2B) and business-to-consumer (B2C) applications, but Ethereum is ideal for transactions between individuals. Due to the above factors, Ethereum is the perfect choice for the peer-to-peer energy trading system as it gives users and developers more flexibility in the type of consensus mechanism that it supports. It also has a broader range of third-party applications and libraries that add to the functionality and ease of use of Ethereum-based smart contracts.

3.5.2 Blockchain Network

The blockchain network layer consists of all the nodes in the network. The blockchain network layer contains components found in the network, physical, virtualization, consensus and incentive layers of the 7-layer blockchain architecture. The main component in this layer is the genesis file which uniquely identifies the network and contains all the rules that govern the network, and this file is created using tools found in the Blockchain protocol layer. All the nodes in the network have a copy of this genesis file. The consensus mechanism to be used by the network is also found at this layer since it is defined on the genesis file. This layer also provides functionality for the nodes in the network to transact amongst each other by sending digital tokens, and a record of all these transactions is stored on this layer. The blockchain network also provides a platform for nodes in the network to deploy as well as call smart contracts. With Ethereum being the chosen blockchain protocol for the peer-to-peer energy trading system, the next step is to select the type of Ethereum network to use. Ethereum has three types of networks, and each one has its own ideal application areas.

1. Ethereum Main Network

The Ethereum main network, which is also known as MainNet, is the biggest public network that uses the Ethereum protocol [88]. It is a permissionless Ethereum network where nodes do not need authorization before they can join the network. The main Ethereum network uses the PoW consensus mechanism, and nodes can participate in mining new blocks, and they get rewarded in Ether which has real-world value in this network [89]. It also costs real money to deploy smart contracts to this network, but it never has any downtime because of its decentralized architecture, which means that the smart contract is always available and cannot be deleted or altered once it has been deployed to the network. This can be a disadvantage in that if it is discovered that a smart contract that has been deployed to this network has an error, that error cannot be changed, and so a new smart contract has to be deployed instead which costs money.

2. Ethereum Test Networks

Ethereum test networks simulate the operations that can be carried out on the main Ethereum network. They allow users to carry out transactions that can be done on the main Ethereum network without using any real Ether. This provides a platform for developers to test smart contracts on an environment that is similar to the final deployment environment without using any real-world money [84]. Instead of making the transactions and contract deployment free, the test networks use a valueless form of Ether to simulate the kind of transaction costs found on the main network. There are many different Ethereum test networks, each with a different network ID, but the most common ones are Ropsten, Kovan, and Rinkeby. Ropsten uses the PoW consensus mechanism, and nodes get valueless Ether by mining blocks on this network. This makes it the test network that best simulates the main Ethereum network. Rinkeby and Kovan use the PoA consensus mechanism, and users request the fake Ether from the test networks' respective web-based applications. This fake Ether is then used to pay transaction costs on the test networks.

3. Ethereum Private Network

The last type of Ethereum blockchain network is a private network. This is a permissioned blockchain based on the Ethereum protocol that allows more flexibility in its rules compared to the other Ethereum networks [90]. Any node can create their own private network and set parameters like which consensus mechanism to use, the name of the network, and the time it takes to create new blocks according to their own needs [91]. Other nodes wishing to join the network will need to be authorized first before they can be a part of the network. This makes it ideal for businesses that want the benefits of blockchain technology without compromising their data privacy by putting it on a network that other people have access to. Since it is based on the Ethereum protocol, it shares similar features with the other types of Ethereum networks in that nodes in the same network can transact with each other, and smart contracts can be deployed to the network. The value of Ether in this type of network is determined by the participants of the network, which gives the participants in the network even more control over the network.

Since some of the transactions in the proposed system might involve low amounts, the best approach is to use a type of network that keeps transaction costs to a minimum. Another factor to take into consideration is the time it takes for transactions to be validated. The proposed system is time-sensitive and so there should not be a big time delay between successive blocks. The only type of Ethereum network that offers flexibility in the configuration of all these parameters is the private network and so it is the network type that will be used for the peer-to-peer energy trading system.

3.5.3 Smart Contract

The smart contract holds the application logic for the entire system. A smart contract is deployed to an address on the blockchain network, and it is called using this address. The smart contract contains methods for the different operations to be carried out in the system, such as listing electricity for sale, purchasing electricity, and being the payment medium for Ether after the transfer of electricity between the buyer and the seller.

3.5.4 Smart Contract Tools

The smart contract tools layer contains components found in the API layer of the blockchain architecture. Ethereum smart contracts are not natively designed to interact with certain external applications, but there are software tools that make this interaction possible. Some of the tools that were used for this purpose are explained below.

1. Truffle

Truffle is an application that provides a development and testing environment for Ethereum smart contracts [92]. Its primary function in the proposed system is to compile and deploy smart contracts to the blockchain network.

2. Web3

Web3 is a set of libraries that enable users to interact with an Ethereum node using various protocols such as IPC, WebSocket, and HTTP. It allows users to initiate transactions between various Ethereum nodes as well as interacting with smart contracts from external applications such as web applications and console applications [83]. It acts as an intermediary between the smart contract and the client-side application.

3. Metamask

Metamask is a browser extension that allows users to use an Ethereum node on a browser. It allows users to add an existing Ethereum account to the extension and carry out transactions from it or from other web-based applications.

3.5.5 Client-side application

The client-side application is in the form of a web application. Users are able to sell as well as buy excess energy. The frontend of the application is populated using data that is stored on the blockchain. In order to view the application, the user's node has to be on the same blockchain network as the smart contract. When a user adds a new listing on the web application, the details of the listing are stored on the smart contract. Whenever

a user intends to purchase energy on the web application, a query is sent to the seller's smart meter to determine if the seller has the amount of energy they are selling.

3.6 Hardware Architecture

This section describes the hardware configuration of the peer-to-peer energy trading system as well as the role of each of the hardware components. All the nodes in the system have a similar structure to make it easier to compare all the nodes' performances. Each node consists of a smart meter, a relay, a battery, and a load that also acts as the system's microcontroller, as illustrated in figure 3.5.

3.6.1 Renewable Energy Storage

This is in the form of a battery, and it is used to represent a fully functional renewable energy system. The battery has a known storage capacity so as to be able to calculate how much energy is remaining at any given point based on how much energy has been used. It has input ports that are used to charge the battery, either from a renewable energy source such as solar or from the batteries of other nodes when energy is purchased from them. It also has multiple output ports that are connected to the input ports of other batteries for when excess energy is sold to other nodes.

3.6.2 Smart Meter

The purpose of the smart meter is to measure the amount of energy that is going into the battery as well as the amount of energy being drawn from the battery by the load. In conjunction with the known capacity of the battery, this is used to determine the amount of energy in the battery at any given time. A smart meter is ideal for this purpose as it can be queried in real-time from an external application. A potential buyer of energy uses this feature to check if the seller has the energy that they have listed on the web application.

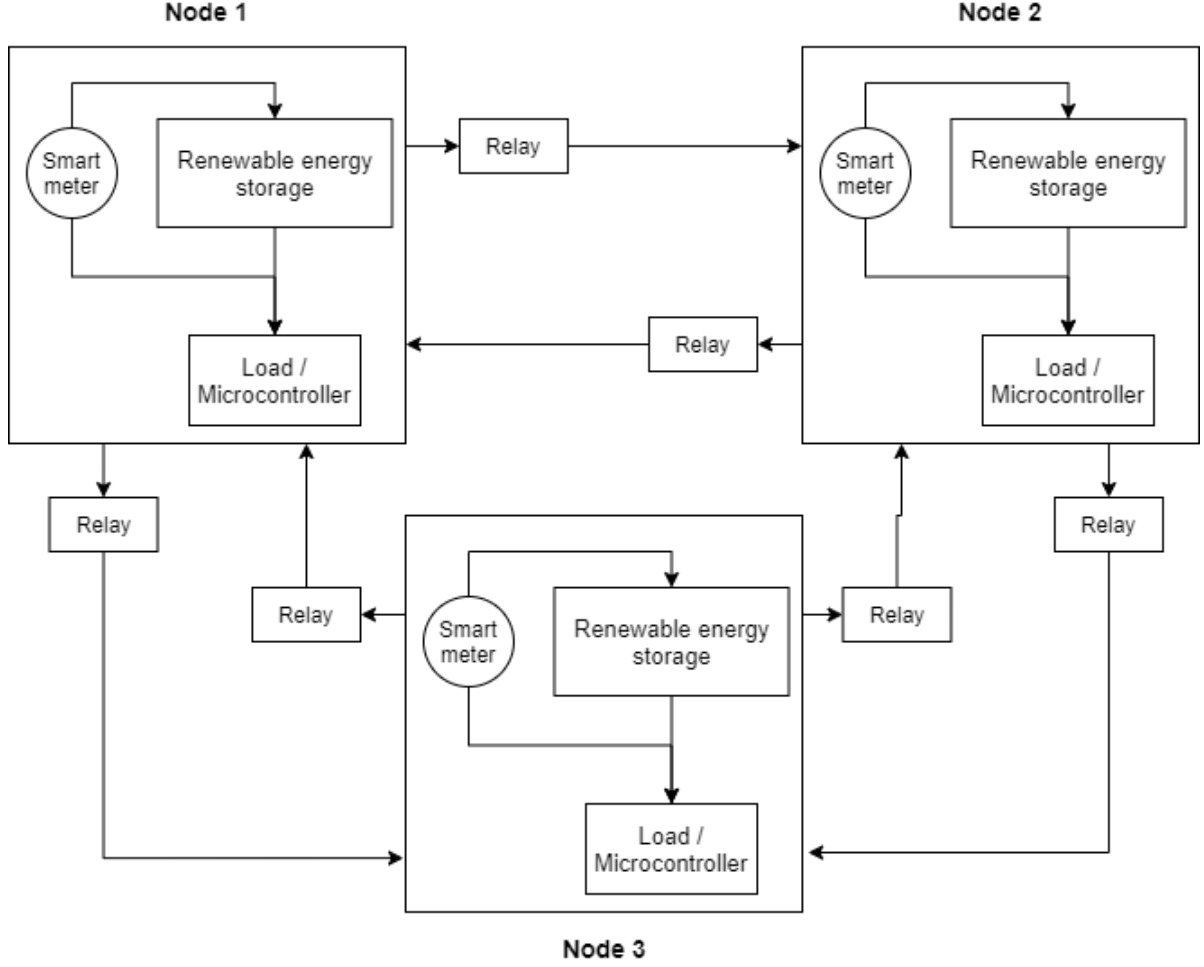


Figure 3.5: Hardware Architecture of the proposed system

3.6.3 Relay

A relay is defined as a switch that is controlled electrically [93]. The primary role of the relay is to allow and prevent the flow of energy in a circuit. It has two states, namely open and closed. If the relay is in the open position, then it means that the circuit is not complete and no current flows through it, but if it is in the closed position, then the circuit allows electric current to flow through it [94]. A relay can be controlled through a signal from a low-power device such as a microcontroller, making it ideal for the proposed system. Each node has a relay for each of the other nodes that it is connected to. This means that a node's battery has multiple output ports that are each connected to another node's battery's input port. A relay is connected between these two points, and by default, it is in the open position, which means that no energy flows through it. If another node purchases energy from this node, a command is sent from the microcontroller for it to

close, and this completes the circuit and allows the energy to flow through it.

3.6.4 Microcontroller

The microcontroller is powered by that node's battery, which means it also acts as a load. The microcontroller is used to control relays by allowing and restricting the flow of energy through them from the battery to the battery of another node. When a buyer purchases energy from a seller, a command is sent to the seller's microcontroller with details about the transaction, such as the seller's identity and the amount of energy that they have purchased. The microcontroller then opens the relay that is connected to the buyer's battery and allows energy to flow from the seller's battery to the buyer's battery. Both the buyer and seller's smart meters will measure the amount of energy that is flowing between the two points, and once the correct amount of energy has been transferred, the microcontroller closes the relay and sends back a command to the web application that the energy has been transferred.

3.7 Data Transfer Protocols

This section details the different protocols that are used to transfer data between different points in the system.

3.7.1 MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol that uses a publish-subscribe model to transfer messages between devices [95]. It is intended for machine to machine connections and is ideal for remote use where the network bandwidth is limited [96]. It runs on TCP/IP. MQTT uses a central broker as shown in Figure 3.6 below. An MQTT broker is an application that receives messages from the publisher and broadcasts them to the clients.

A publisher publishes messages on a particular topic to a broker. The purpose of the broker is to categorize the messages according to the topic and then distribute the messages to the subscribers. For a client to receive these messages, they have to subscribe to that particular topic on the same broker. The broker discards the message after

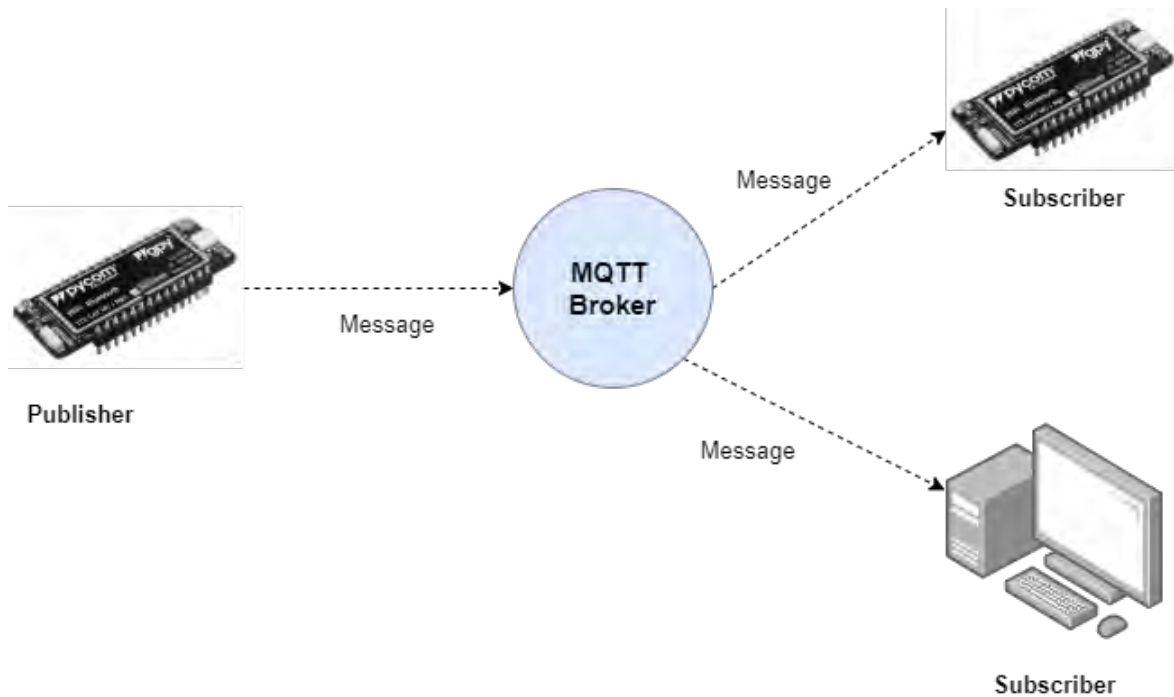


Figure 3.6: MQTT Architecture

distributing it to the subscribers unless the publisher specifies that it should be stored [97]. The publisher and the client are not directly connected and only communicate through the broker. MQTT is used in the peer-to-peer energy trading system to transmit data from the smart meter to a database. It is more lightweight than other protocols that could have been used for this purpose such as HTTP which makes it more ideal. MQTT has a smaller header and message and this enables the smart meter to send data to the database more frequently without using a lot of bandwidth.

3.7.2 RLPx

The RLPx protocol is a TCP-based transport protocol that is used by the Ethereum protocol to govern how peers communicate with each other [98]. When two peers on an Ethereum network attempt to communicate for the first time, they implement a two-phase handshake. In the first phase, the peers exchange cryptographic information which they use to encrypt and authenticate subsequent messages. In the second phase of the handshake, the peers exchange their capabilities such as the sub-protocol that they support so that they use that protocol for application-level communication [99].

3.8 Conclusion

In this chapter, two design approaches were assessed and a choice was made on the most appropriate one for the design of the proposed system. The proposed system was divided into hardware components and software components and the architectures of both parts were proposed. The blockchain protocol to be used for the implementation of the solution was selected and the hardware components that will be required by the system were also identified.

Chapter 4

Private Blockchain Network

This chapter provides the design of a private blockchain network for the peer-to-peer energy trading system with multiple nodes that can transact with each other. The private blockchain network is created using Ethereum as it is the blockchain protocol that was chosen for this research. This chapter also defines some key technologies behind Ethereum that explain how transactions will be handled in the proposed system.

4.1 Ethereum Virtual Machine

Ethereum is not only considered a distributed ledger but also a distributed state machine due to its ability to support smart contracts in addition to its functionality of distributed transactions [100]. A state in a blockchain is a data structure containing a list of all the accounts on the blockchain and their balances. Ethereum's state also holds a machine state which can change from one block to another, and it follows a set of pre-defined rules and can execute machine code [101].

The rules of changing the state from one block to the next are defined by the Ethereum Virtual Machine (EVM). The EVM is a single canonical computer whose state is agreed upon by every node on the Ethereum network. Every participant on the network keeps a copy of this computer state and can broadcast a request to the other network participants for the EVM to carry out a computation. The other participants then verify, validate and execute the requested computation, and these requests are known as transaction requests. Once the computation has been carried out, this results in a change of state in the EVM, and this change is reflected throughout the network.

The EVM is leveraged in the proposed system to deploy a smart contract to the blockchain. Every transaction that is carried out in the energy trading system changes the state of the EVM and every participant will be required to have an updated EVM state in order to continue participating in the system. This is to prevent participants from spending the same funds more than once due to having an outdated EVM state. The Ethereum protocol automatically checks if a participant has an updated EVM state before they can transact.

4.2 Ethereum Node

An Ethereum node is defined as a computer that is running a piece of software known as an Ethereum client [102]. An Ethereum client is an implementation of the Ethereum protocol in a particular programming language that verifies all the transactions in a block and is responsible for the security of the network as well as maintaining data accuracy in the blockchain. There are three types of Ethereum nodes: a full node, a light node, and an archive node.

A full node stores the full blockchain data, and it participates in block validation. A full node also verifies all the blocks as they are added to the blockchain, and it provides data to the network on request [103]. A light node only stores the block headers and requests everything else from full nodes when needed. A light node can verify if the data in a block is valid by checking the block headers' state roots. Light nodes are useful for devices with low storage capacities, such as mobile phones, as the full Ethereum data consumes a lot of space [104]. An archive node stores everything stored on the full node, but it also builds an archive of the Ethereum EVM's historical states. This is useful for querying historical data, such as querying a particular address's account balance at a specific point in time. Archive nodes require a lot of storage space as the current Ethereum mainnet archive is more than 5 Terabytes in size and is constantly growing [102].

The peer-to-peer energy trading system uses full nodes throughout the blockchain because full nodes meet the system requirements such as the ability to validate blocks which

means any of the nodes on the blockchain can be selected to validate blocks. Full nodes also keep a full copy of the blockchain which makes tempering with the blockchain very difficult and improbable. Full nodes also do not have the excessive storage requirements of archive nodes as they only keep the updated copy of the blockchain and discard the previous versions. This reduces the hardware requirements for the peer-to-peer energy trading system without limiting its functionality.

4.3 Ethereum Client

An Ethereum client is an implementation of the Ethereum protocol, and it allows nodes to participate on the Ethereum network. A client comes preconfigured with all the tools necessary for the node to transact as well as validate other transactions. Figure 4.1 shows the features of an Ethereum client. It consists of the consensus mechanism such as the PoW as well as the EVM. The Ethereum client also contains the current state of the EVM as well as the protocols for broadcasting transactions to the other nodes on the network. The last feature of an Ethereum client is known as a TX Mempool. This is a dynamic memory area that contains pending transactions before they are included in a block.

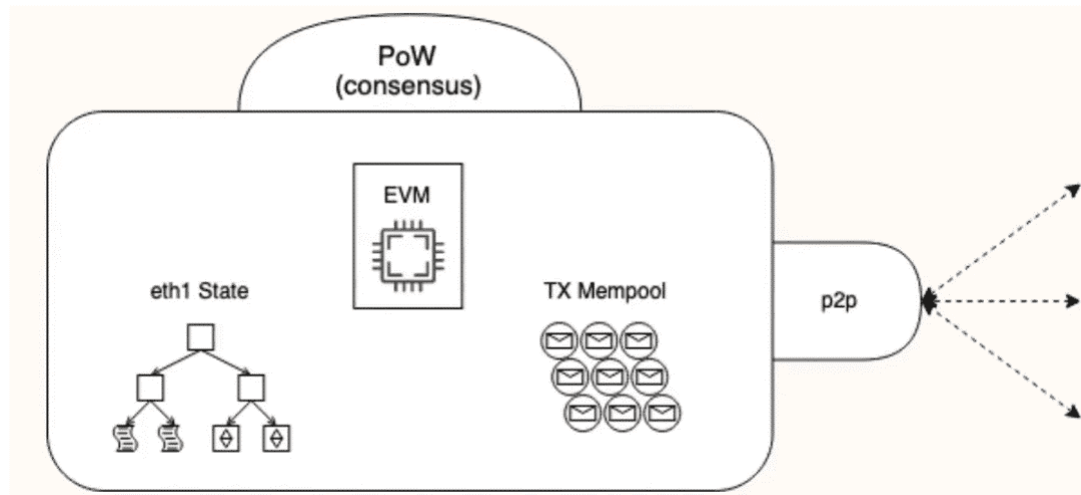


Figure 4.1: Features of an Ethereum client[102]

There are currently many clients that implement the Ethereum protocol as Ethereum is an open-source platform [102]. Each of these clients has its own advantages, and they

support different operating systems. According to Ethereum, the goal is to make the network diverse by having many different clients to reduce any single points of failure. The three most popular clients are Geth, OpenEthereum, and Nethermind. The three are evaluated below before a choice on the appropriate client for the energy trading system is made.

1. Geth

Geth, which is short for Go Ethereum, is one of the oldest implementations of the Ethereum protocol [105]. It is written in the Go programming language and supports Linux, Windows, and macOS. Geth is open-source and is the most widely used Ethereum client, and developers have developed a lot of tools to automate some tasks that can be carried out in Geth.

2. OpenEthereum

OpenEthereum is an Ethereum client written using the Rust programming language. It aims to be the lightest, fastest, and most secure Ethereum implementation [106]. OpenEthereum has features for easy customization, and it has the most lightweight storage and memory footprint.

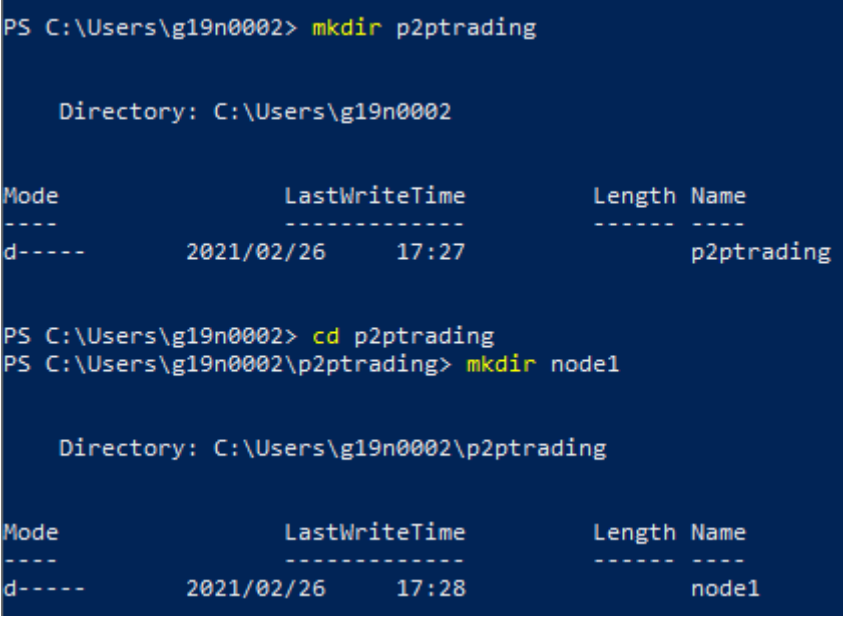
3. Nethermind

Nethermind is a C# .NET-based Ethereum client. It runs on all the major platforms, including ARM, and performs well on virtual machines [107]. Nethermind also has an active online community and provides constant support for its premium customers.

Geth has the largest online community out of the three options and also has the most applications to make some processes easier and so it is the client chosen to set up the nodes for all the participants in the proposed system. Geth is also lightweight enough to run on low-specification computers.

4.4 Setting up an Ethereum Node

This section outlines the steps that were taken to create an Ethereum node as well as an account on that node. The one prerequisite is an Ethereum client, and Geth was the chosen client for this purpose. Once Geth had been installed, the first step required the creation of a working directory where the Ethereum accounts and the blockchain data were to be stored.



```
PS C:\Users\g19n0002> mkdir p2ptrading

Directory: C:\Users\g19n0002

Mode                LastWriteTime         Length Name
----                -
d-----          2021/02/26   17:27                p2ptrading

PS C:\Users\g19n0002> cd p2ptrading
PS C:\Users\g19n0002\p2ptrading> mkdir node1

Directory: C:\Users\g19n0002\p2ptrading

Mode                LastWriteTime         Length Name
----                -
d-----          2021/02/26   17:28                node1
```

Figure 4.2: Creating working directories

Figure 4.2 shows the creation of the working directory called p2ptrading. This directory stores all the data from the blockchain for this node such as the genesis file. The other directory created is called node1, and it holds the encrypted private keys for all the accounts that will be created on this node as well as the blockchain ledger itself. After this, the next step was to create the accounts for the node. An Ethereum account is an entity for storing Ether, and it has a balance. It is identified by an address known as an ETH address or an ERC20 address. An ETH address is prefixed by 0x, which is followed by forty alphanumeric characters. A single node can have multiple accounts. To create an account, Geth was used, as shown in Figure 4.3. These accounts are where the participants of the energy trading system keep their cryptocurrency that they use to transact within the system.

```
PS C:\Users\g19n0002\p2ptrading> geth --datadir node1/ account new
INFO [02-26|17:30:26.807] Maximum peer count          ETH=50 LFS=0 total=50
Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key:  0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29
Path of the secret key file: node1\keystore\UTC--2021-02-26T15-30-41.330819800Z--e91d849375e8ba9c0b83b2c8006dd4ec3ca71a29

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!
```

Figure 4.3: Creating a new Ethereum account

The `datadir` flag is required when creating an account with Geth, and it specifies the directory where the private key of the account must be stored, and in this case, it is the `node1` directory created earlier. When creating an account, a user must set a password as Geth does not store the Ethereum private key in plain text but stores it in an encrypted JSON file. The private key is encrypted using a 128-bit AES encryption algorithm. A user needs to enter this password whenever they want to use their account, such as when transacting with other nodes. After entering their preferred password, Geth generates two things. The first thing that is generated is the public key which is the ETH address. A user can share their public address with other users as this is the address they use if they want to transact with them. The second key that is generated is the encrypted private key, and a user should never share it with anyone. The encryption adds an extra security layer as someone would need both the private key JSON file as well as the password for them to gain access to another user's account, but it also means that a user can lose access to their account forever if they lose their password as there is no way to recover it.

Once a node has an account, then it can join Ethereum networks such as the Mainnet and transact with other nodes on that network. A single account can be used on multiple networks, and it will have different account balances on each network. To join a network, a node just needs to have the genesis file of the network they wish to join. Geth comes preconfigured with the genesis file of the Mainnet as well as other popular test networks, and a user can join these. Geth also allows users to create their own Ethereum networks and set their own rules for the network.

4.5 Setting up a Private Ethereum Network

A private Ethereum network was selected as the most ideal type of network for the peer-to-peer energy trading system as it gives full control of the blockchain to its participants while maintaining the confidentiality of the data from outsiders. This allows the participants to attach their own value of the cryptocurrency within the network. Ethereum allows users to create their own private blockchain networks and set their own rules about how the network should operate. An Ethereum network is governed and identified through a genesis file, and so to create a network for the energy trading system, a genesis file needed to be created. An Ethereum genesis file follows certain formatting rules, which, if not followed correctly, can prevent the network from functioning properly.

To make it easier to create a new network, Geth provides a tool called Puppeth that takes user input and generates the genesis file. Puppeth is a command-line-based Ethereum private network manager. It provides tools for analysing an Ethereum network by tracking details about each block added to the blockchain and all the nodes on the network. It also helps users generate and manage a genesis file without worrying about the formatting rules. Puppeth presents a series of prompts to the user about their preferences for the configuration of the network, and the responses to these prompts are used to create the network and generate the genesis file. Puppeth comes pre-installed in Geth, and so to start it, the keyword `puppeth` was entered into the console window as shown in figure 4.4. This starts up the Puppeth console application.

After starting up Puppeth, the first thing that was required was to enter the network name for the blockchain network that was to be created. The network name is not as important as the network ID, but Puppeth uses it for the purposes of identifying the network for analysis. The next prompt was to decide what was to be done to the network entered in the previous step. The options range from configuring a new genesis file to viewing the statistics of the network. After selecting the option to configure a new genesis file, two more options were presented: creating a new genesis file from scratch or importing a genesis file that was created elsewhere. Once this step was completed, the configuration of the genesis file could begin. The first and most important design question

```
PS C:\Users\g19n0002\p2ptrading> puppeth
+-----+
| Welcome to puppeth, your Ethereum private network manager |
|
| This tool lets you create a new Ethereum network down to |
| the genesis block, bootnodes, miners and ethstats servers |
| without the hassle that it would normally entail.         |
|
| Puppeth uses SSH to dial in to remote servers, and builds |
| its network components out of Docker containers using the |
| docker-compose toolset.                                   |
+-----+

Please specify a network name to administer (no spaces, hyphens or capital letters please)
> p2ptrading

Sweet, you can set this via --network=p2ptrading next time!

@[32mINFO @[0m[03-01|10:14:26.420] Administering Ethereum network           @[32mname@[0m=p2ptrading
@[33mWARN @[0m[03-01|10:14:26.447] No previous configurations found         @[33mpath@[0m=.puppeth\p2ptrading

What would you like to do? (default = stats)
 1. Show network stats
 2. Configure new genesis
 3. Track new remote server
 4. Deploy network components
> 2

What would you like to do? (default = create)
 1. Create new genesis from scratch
 2. Import already existing genesis
> 1
```

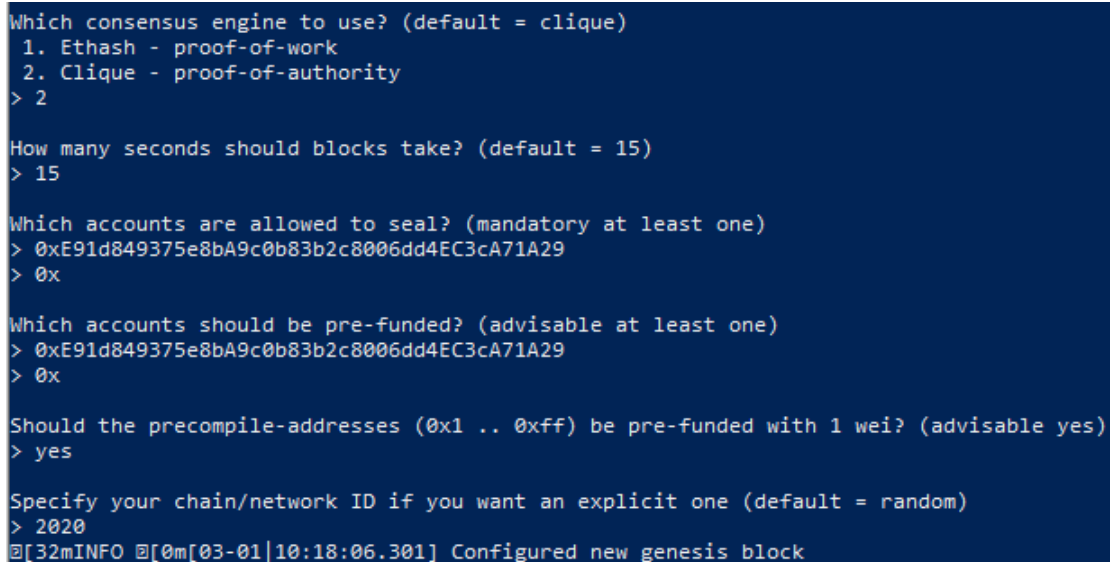
Figure 4.4: Starting up Puppeth and creating a new genesis file

was the appropriate consensus engine to use. Geth comes with two different consensus engines that both use different consensus mechanisms, namely Ethash, which is based on the proof of work consensus mechanism, and Clique, which uses proof of authority. This was one of the most important design considerations for the project as it affects a lot of system functions and requirements.

A PoA-based consensus engine such as Clique is better suited for the peer-to-peer energy trading platform’s requirements compared to a PoW-based engine like Ethash for a number of reasons. The first factor to consider is the computing power required by both consensus mechanisms. In Ethash, significant computing power is required as the validation of blocks involves mining. The minimum recommended standard is a desktop with a graphics card with at least 4GB of RAM. This is a significant amount of RAM, and it will increase the cost of the system as every participating node will need to meet that requirement if they are to validate blocks on the network. On the other hand, Clique does not require any mining as only a few pre-authorised nodes can validate blocks. The Clique consensus engine is so efficient that it can be run on nodes with only 256MB of GPU memory. This means that devices such as mobile phones and Raspberry Pis can validate blocks on blockchain networks based on the Clique consensus engine.

The next factor to consider is the security of both consensus engines against malicious users. Ethash relies on the need for high computing capabilities to guard against malicious users potentially gaining control of the network. This works very well on large public networks where it is highly unlikely that any one entity can have more than half of the entire network's computing power. It, however, does not work as well on smaller networks where each node is using a regular desktop computer because it is possible for one of the participant's computers to be more powerful than the sum total computing power of all the other participants. This will allow that node to gain control of the network and manipulate blocks in their favour. With Clique, the validators are known by all the participants in the network. The requirements for one to become a validator reduce the incentive for a validator to act maliciously. If the network has a significant number of validators, then a node would need to collude with more than half of them to have a chance of manipulating blocks [18].

Due to these factors, a consensus engine that uses the PoA consensus mechanism is ideal for this system and so Clique was selected as the consensus engine to be used.



```
Which consensus engine to use? (default = clique)
1. Ethash - proof-of-work
2. Clique - proof-of-authority
> 2

How many seconds should blocks take? (default = 15)
> 15

Which accounts are allowed to seal? (mandatory at least one)
> 0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29
> 0x

Which accounts should be pre-funded? (advisable at least one)
> 0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29
> 0x

Should the precompile-addresses (0x1 .. 0xff) be pre-funded with 1 wei? (advisable yes)
> yes

Specify your chain/network ID if you want an explicit one (default = random)
> 2020
[32mINFO [0m[03-01|10:18:06.301] Configured new genesis block
```

Figure 4.5: Selecting the consensus engine using Puppeth

After choosing Clique as the preferred consensus mechanism, the next prompt was to select the time it takes for new blocks to be created. In PoA-based blockchains, a new block is added to the blockchain at fixed intervals regardless of whether or not there have

been any transactions within that period. The most important factor when choosing the time it takes to create new blocks is the sensitivity of the intended application area to any time delay. If the blockchain is being used for time-critical applications, then this time should be kept low. The peer-to-peer energy trading system is sensitive to a big time delay as a user might have very little energy left in their battery when making the purchase, and so the default time of 15 seconds is maintained. This means that a new block will be added to the blockchain every 15 seconds, and a transaction can take up to a maximum of 15 seconds before it is validated. The transfer of energy from a seller to a buyer can only commence once the block containing that transaction has been validated.

The next step was to select the accounts that would be the initial sealer nodes for the network. The sealer nodes are responsible for validating new blocks in the network, and at least one sealer node has to be defined when creating the network. Since there was only one node on the network, that node was chosen as the initial sealer node. The rest of the prompts were about pre-funding the account. In PoW-based networks, this is not necessary as the accounts can mine for Ether, but in a PoA-based network, the total amount of Ether that can circulate on the network is set when creating it. The account that was to be pre-funded with this Ether was specified on the prompt, and this account appears on the genesis file.

The last prompt required the setting of a network ID. This is what is used by other nodes to identify the network, and it is required before a node joins the network. If the blockchain network is to be broadcast over the internet, then a unique network ID must be set. Every public Ethereum network has a unique network ID, and the network ID for the Mainnet is 1. Our blockchain network is not public, and so any network ID is suitable, and in this case, the chosen ID was 2020. This was the last step to generate a genesis file for the blockchain, and after this, the genesis file was saved to the current directory.


```

1  {
2    "config": {
3      "chainId": 2020,
4      "homesteadBlock": 0,
5      "eip150Block": 0,
6      "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
7      "eip155Block": 0,
8      "eip158Block": 0,
9      "byzantiumBlock": 0,
10     "constantinopleBlock": 0,
11     "petersburgBlock": 0,
12     "istanbulBlock": 0,
13     "clique": {
14       "period": 15,
15       "epoch": 30000
16     }
17   },
18   "nonce": "0x0",
19   "timestamp": "0x603ca26d",
20   "extraData": "0x0000000000000000000000000000000000000000000000000000000000000000e91d849375e8ba9c0b83b2c8006dd4ec3ca71a2900",
21   "gasLimit": "0x47b760",
22   "difficulty": "0x1",
23   "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
24   "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",
25   "alloc": {
26     "e91d849375e8ba9c0b83b2c8006dd4ec3ca71a29": {
27       "balance": "5000"
28     }
29   },
30   "number": "0x0",
31   "gasUsed": "0x0",
32   "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
33 }

```

Figure 4.6: The generated genesis file

4.6 Genesis File

Figure 4.6 shows the generated genesis file. The file is in JSON format and has the same name as the blockchain network. The genesis file contains some important information that governs how the network will function. The blocks found on the genesis file are explained below.

1. Config

The config block contains the core blockchain settings and parameters that control the operations of the network. It also contains properties relating to the version of Ethereum to use. There are many different versions of Ethereum, each with a small difference from another version, and these versions are referred to as forks. HomesteadBlock, eip158Block, eip155Block, byzantiumBlock, constantinopleBlock, and istanbulBlock all refer to different forks of Ethereum, and they are all set to zero as each of their features are used in the proposed blockchain network.

2. ChainId

The chain ID refers to the network ID that was specified when creating the network. It is used to identify the blockchain network. Nodes wishing to join the blockchain

network for the peer-to-peer energy trading system will need to use 2020 as the chain ID as it is what was specified when the network was created. The chain ID also acts as a security mechanism against replay attacks. A replay attack is when a malicious user intercepts network traffic and then sends it to its original destination but acting as the original sender. The chain ID is included in the transaction signature, which prevents replay attacks by making it impossible for anyone to decipher the network traffic without the correct chain ID.

3. Clique

The Clique block specifies the consensus mechanism that the blockchain network will use. This block contains two parameters. The period refers to the time between two successive blocks, and this was set when the network was being created. This value cannot be changed once the network is started. The other parameter is called the epoch, and this is the number of blocks it will take for the network to create a checkpoint and reset any pending block.

4. Nonce and MixHash

These values are primarily used in PoW-based blockchains to verify that a block has been mined. The nonce is a 64-bit value, and it is combined with the 256-bit mixHash to check that a sufficient amount of cryptographic computation has been done on the block. Since no computation is carried out to validate a block in a PoA-based blockchain, both values were set to zero.

5. ExtraData

This field is used to specify the initial sealer nodes on the network. PoW-based blockchain networks do not have this field in their genesis files as there are no sealer nodes but PoA-based blockchain networks do and so this is reflected in the genesis file.

6. GasLimit

Gas in Ethereum refers to the cost associated with performing a transaction in an Ethereum network. The price is determined by the supply and demand of

computational power as well as the amount of computation that a transaction requires. The `gasLimit` is the maximum amount of gas that can be used for each block. This was set to a high value as the blockchain network will be used for smart contracts that might require a lot of computations. It is also possible to set that the `gasLimit` should increase every time a new block is added to the blockchain which would mean that the older a blockchain network is, the more complex computations it will be able to carry out.

7. Difficulty

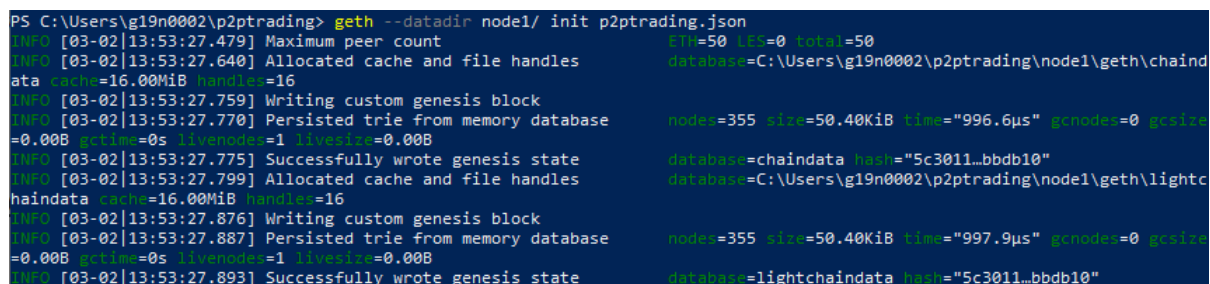
This refers to the processing power required to mine blocks in the network. This is only applicable to PoW-based blockchains, and the higher this value, the more computational power is required to mine. In a PoA-based blockchain, this value is always set to 1 by default, which means there is very little computational power required to validate blocks.

8. Alloc

The `alloc` block specifies the accounts that should be pre-funded when the network is created as well as the amount of Ether that the account should have.

4.7 Starting the Blockchain Network

This section outlines the steps that were taken to start the blockchain network that was created previously as well as the test that was carried out to check if the network was functioning properly.

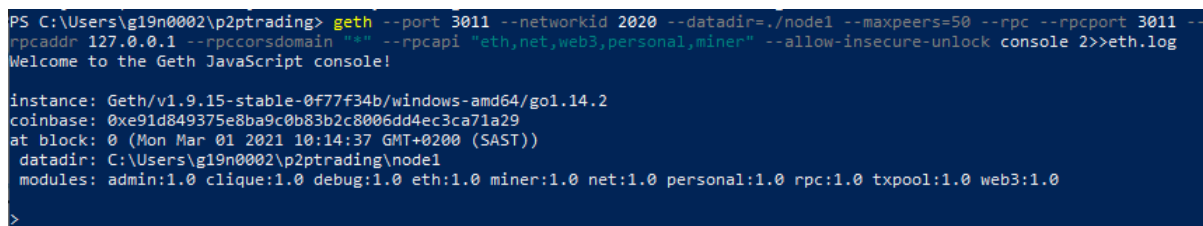


```
PS C:\Users\g19n0002\p2ptrading> geth --datadir node1/ init p2ptrading.json
[INFO] [03-02|13:53:27.479] Maximum peer count           ETH=50 (FS=0 total)=50
[INFO] [03-02|13:53:27.640] Allocated cache and file handles      database=C:\Users\g19n0002\p2ptrading\node1\geth\chaindata
ata cache=16.00MiB handles=16
[INFO] [03-02|13:53:27.759] Writing custom genesis block
[INFO] [03-02|13:53:27.770] Persisted trie from memory database    nodes=355 size=50.40KiB time="996.6µs" gcnodes=0 gcsiz
=0.00B gctime=0s livenodes=1 liveness=0.00B
[INFO] [03-02|13:53:27.775] Successfully wrote genesis state       database=chaindata hash="5c3011...bbdb10"
[INFO] [03-02|13:53:27.799] Allocated cache and file handles      database=C:\Users\g19n0002\p2ptrading\node1\geth\lightc
haindata cache=16.00MiB handles=16
[INFO] [03-02|13:53:27.876] Writing custom genesis block
[INFO] [03-02|13:53:27.887] Persisted trie from memory database    nodes=355 size=50.40KiB time="997.9µs" gcnodes=0 gcsiz
=0.00B gctime=0s livenodes=1 liveness=0.00B
[INFO] [03-02|13:53:27.893] Successfully wrote genesis state       database=lightchaindata hash="5c3011...bbdb10"
```

Figure 4.7: Initializing the genesis file

The first step was to initialize the genesis file, as shown in figure 4.7. The purpose

of initializing the genesis file was to create databases that store some of the data related to the blockchain, such as the list of nodes on the blockchain as well as the databases to store the ledger of transactions. Every node wishing to join the blockchain has to initialize the same genesis file. Only the folder in which the blockchain data will be stored has to be specified by the user as one of the parameters for the initialization process. The next step was to start the Geth console application that allows users to interact with the blockchain, transact and carry out other operations on the network. Figure 4.8 shows the command that was used to start the Geth console application.



```
PS C:\Users\gl9n0002\p2ptrading> geth --port 3011 --networkid 2020 --datadir=./node1 --maxpeers=50 --rpc --rpcport 3011 --  
rpcaddr 127.0.0.1 --rpccorsdomain "*" --rpcapi "eth,net,web3,personal,miner" --allow-insecure-unlock console 2>>eth.log  
Welcome to the Geth JavaScript console!  
  
instance: Geth/v1.9.15-stable-0f77f34b/windows-amd64/go1.14.2  
coinbase: 0xe91d849375e8ba9c0b83b2c8006dd4ec3ca71a29  
at block: 0 (Mon Mar 01 2021 10:14:37 GMT+0200 (SAST))  
datadir: C:\Users\gl9n0002\p2ptrading\node1  
modules: admin:1.0 clique:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0  
>
```

Figure 4.8: Starting the Geth console

To start the Geth console application, a number of parameters had to be set to allow more things to be done on the console application. These parameters were set using flags when starting the console application, and these flags are explained below.

1. Port

The port flag specifies the network port that the node must use to listen to broadcasts from other nodes. These broadcasts include any communication between nodes, such as transaction broadcasts and requests to add or remove a sealer node.

2. Network ID

The network ID is the identifier of the network, and every node that wishes to join the network will need to start the Geth console using the correct network ID.

3. Datadir

This is the data directory where the blockchain data including the ledger of transactions is stored. This folder must be the same as the one that was used to initialize the genesis file.

4. RPC

This flag is for starting the HTTP JSON-RPC. The HTTP JSON-RPC is a remote procedure call protocol that allows a program to request a service from a program located in another computer in the same network without having to understand the network configuration first [108].

5. RPC CORS Domain

This flag lists the domains from which the console application can accept cross-origin requests. By default, JavaScript does not accept cross-origin requests due to the same-origin policy [109]. This is to prevent scripts from accessing malicious content. In the proposed blockchain network, this flag is set to accept requests from all other domains as it is possible that some requests may appear to originate from the same source, which may hinder some operations on the blockchain if the same-origin policy is not removed.

6. RPC API

This flag allows users to list Application Programming Interfaces (API) that they want to have access to on the console application to make some functions easier to carry out. The APIs listed include `net`, which is an Ethereum package that allows users to view the node's network properties as well as the `web3` API, which allows users to initiate transactions on the console application. These APIs are useful in the peer-to-peer energy trading system as they make some tasks such as checking account balances or paying for electricity much easier.

7. Console

Geth comes with a JavaScript Read, Evaluate, and Print Loop (REPL) console application that enables the user to carry out transactions on the network. The logs of all the actions that are carried out on the blockchain network are stored in the `eth.log` file. The log file records everything that happens on the blockchain, such as transactions and smart contract calls. Figure 4.9 shows the `eth.log` file after the blockchain network is started.

```

1  geth : INFO [03-02|14:01:39.174] Maximum peer count          ETH=50 LES=0 total=50
2  At line:1 char:1
3  + geth --port 3011 --networkid 2020 --datadir=./node1 --maxpeers=50 --r ...
4  + ~~~~~
5  + CategoryInfo          : NotSpecified: (INFO [03-02|14:... LES=0 total=50:String) [], RemoteException
6  + FullyQualifiedErrorId : NativeCommandError
7
8  WARN [03-02|14:01:39.175] The flag --rpc is deprecated and will be removed in the future, please use --http
9  WARN [03-02|14:01:39.181] The flag --rpcaddr is deprecated and will be removed in the future, please use --http.addr
10 WARN [03-02|14:01:39.181] The flag --rpcport is deprecated and will be removed in the future, please use --http.port
11 WARN [03-02|14:01:39.181] The flag --rpcorsdomain is deprecated and will be removed in the future, please use
12 --http.corsdomain
13 WARN [03-02|14:01:39.181] The flag --rpcapi is deprecated and will be removed in the future, please use --http.api
14 INFO [03-02|14:01:39.251] Starting peer-to-peer node
15 instance=Geth/v1.9.15-stable-0f77f34b/windows-amd64/go1.14.2
16 INFO [03-02|14:01:39.268] Allocated trie memory caches        clean=256.00MiB dirty=256.00MiB
17 INFO [03-02|14:01:39.269] Allocated cache and file handles
18 database=C:\Users\g19n0002\p2ptrading\node1\geth\chaindata cache=512.00MiB handles=8192
19 INFO [03-02|14:01:39.653] Opened ancient database
20 database=C:\Users\g19n0002\p2ptrading\node1\geth\chaindata\ancient
21 INFO [03-02|14:01:39.654] Initialised chain configuration      config="{ChainID: 2020 Homestead: 0 DAO: <nil>
22 DAOsupport: false EIP150: 0 EIP155: 0 EIP158: 0 Byzantium: 0 Constantinople: 0 Petersburg: 0 Istanbul: 0, Muir Glacier:
23 <nil>, YOLO v1: <nil>, Engine: clique}"
24 INFO [03-02|14:01:39.671] Initialising Ethereum protocol      versions="[65 64 63]" network=2020 dbversion=<nil>
25 WARN [03-02|14:01:39.671] Upgrade blockchain database version  from=<nil> to=7
26 INFO [03-02|14:01:39.673] Loaded most recent local header      number=0 hash="5c3011fC#bbdb10" td=1 age=1d3h47m
27 INFO [03-02|14:01:39.682] Loaded most recent local full block  number=0 hash="5c3011fC#bbdb10" td=1 age=1d3h47m
28 INFO [03-02|14:01:39.682] Loaded most recent local fast block  number=0 hash="5c3011fC#bbdb10" td=1 age=1d3h47m
29 INFO [03-02|14:01:39.684] Regenerated local transaction journal transactions=0 accounts=0
30 INFO [03-02|14:01:39.731] Allocated fast sync bloom           size=512.00MiB
31 INFO [03-02|14:01:39.737] Initialized fast sync bloom          items=355 errorrate=0.000 elapsed=5.019ms
32 INFO [03-02|14:01:39.739] Stored checkpoint snapshot to disk   number=0 hash="5c3011fC#bbdb10"
33 INFO [03-02|14:01:39.840] New local node record               seq=1 id=9eefcf39581014eb ip=127.0.0.1 udp=3011
34 tcp=3011
35 INFO [03-02|14:01:39.843] Started P2P networking               self=enode://d01c38ef40a08e6916d61e1cea4a43fff8fb121fd
36 1bed7598c48b8e2a69e471cbd7edfe73b48e4b49135bcc57e80fd951e0695cfce5921ae4a79b5abcfce83a5@127.0.0.1:3011
37 INFO [03-02|14:01:39.877] IPC endpoint opened                  url=\\.\pipe\geth.ipc
38 INFO [03-02|14:01:39.877] HTTP endpoint opened                 url=http://127.0.0.1:3011/ cors=* vhosts=localhost
39 INFO [03-02|14:01:39.983] Etherbase automatically configured  address=0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29
40 INFO [03-02|14:01:41.971] New local node record               seq=2 id=9eefcf39581014eb ip=146.231.88.74 udp=3011
41 tcp=3011
42 INFO [03-02|14:01:50.698] Looking for peers                    peercount=0 tried=91 static=0
43 INFO [03-02|14:02:01.135] Looking for peers                    peercount=0 tried=87 static=0

```

Figure 4.9: eth.log file after the Geth console is started

4.8 Geth Console

Once the Geth console had been started, the first thing that had to be done was to check if the network was up and functioning correctly. A simple way to do this is to check the account balance of the node that created the network. The initial account balance was known as it was defined in the genesis file, and so if the result of the test was to be different from the one in the genesis file, then that means there would have been an error in the configuration of the network. In this instance, the account balance should return 5000.

Before carrying out any operation using an account in Ethereum, the account had to be unlocked using the password that was set when the account was created, as illustrated in figure 4.10. This is because the private key that is required to sign any transaction is stored in an encrypted format and so has to be decrypted first. To unlock an account, the `unlockAccount` function of the personal API is used. The function takes two arguments in the form of the address of the account to be unlocked and the password. The function

returns a Boolean value of true if the operation was successful or false if it was not. After the account was successfully unlocked, the account balance was checked using the eth API's getBalance function. The operation returned a value of 5000 Ether which matched the value in the genesis file. This showed that the network was operating correctly, and other nodes could now be added to the network.

```
> personal.unlockAccount("0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29","tyron123")
true
> eth.getBalance("0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29")
5000
>
```

Figure 4.10: Unlocking account and checking account balance

4.8.1 Adding Other Nodes

When the network was up and running, the next step was to add other nodes. This is because the peer-to-peer energy trading system requires each participant to be represented by a node on the blockchain. The first node can check for other nodes on the network using the command shown in figure 4.11. The request returned an empty array as there were no other nodes on the network at that point.

```
> admin.peers
[]
>
```

Figure 4.11: Checking for other nodes on the network

The main prerequisite for a node to join the private network is a copy of the genesis file. The node that created the network had to distribute the genesis file to all the nodes that intended to join the network. The other nodes had to initialize the genesis file in the same way as the first node. The other requirement for a node to join a private Ethereum network is the enode of at least one sealer node on the network. The enode is a unique identifier allocated to every Ethereum node, which consists of a hexadecimal node ID, the node's IP address, and its listening port that was defined when starting up the Geth console. The enode of the first node is shown in figure 4.12.

```
> admin.nodeInfo
{
  enode: "enode://39bf2c683442440e28cc4e6e0a59fb20b1247cbf9b0c2f9607f8731f7f5146e5eae598d87e18cf15678e850ba64e30cc19af8c083ed692ea5956adc8cc28343c@146.231.88.74:3011",
```

Figure 4.12: enode of the first node

Once a node has these two requirements, they have the necessary permission to join the network. To join a network after starting up the Geth console with the appropriate genesis file, a node has to add the sealer node as a peer, as shown in figure 4.13.

```
> admin.addPeer("enode://39bf2c683442440e28cc4e6e0a59fb20b1247cbf9b0c2f9607f8731f7f5146e5eae598d87e18cf15678e850ba64e30cc19af8c083ed692ea5956adc8cc28343c@146.231.88.74:3011?discport=0")
true
> admin.peers
[[{
  caps: ["eth/63", "eth/64", "eth/65"],
  enode: "enode://39bf2c683442440e28cc4e6e0a59fb20b1247cbf9b0c2f9607f8731f7f5146e5eae598d87e18cf15678e850ba64e30cc19af8c083ed692ea5956adc8cc28343c@146.231.88.74:3011?discport=0",
  id: "37c136347e3c669f58f0a5527e2adad08b2ba21d445937f41c2849f8113e04",
  name: "Geth/v1.9.15-stable-0f77f34b/windows-amd64/go1.14.2",
  network: {
    inbound: false,
    localAddress: "146.231.88.74:52055",
    remoteAddress: "146.231.88.74:3011",
    static: true,
    trusted: false
  },
  protocols: {
    eth: {
      difficulty: 37,
      head: "0x90c220719144c0e32e9247ff7b3bcb34f3f48ae90650c928e8c83ed8125d7fbd",
      version: 65
    }
  }
}]
```

Figure 4.13: The second node joining the network using the enode of the first node

The function `addPeer` returns a Boolean value of true or false depending on whether or not the operation was successful. When the same query that was run before to check for other nodes was run again, it returned an array with the details of the other node on the network. This query can be run on both nodes, as shown in Figure 4.13 for the second node as well as in Figure 4.14 for the first node, and it should return the information of the other nodes on the network. If this query returns the details of the other nodes on the network, that means the new node has successfully joined the network and can now transact with other nodes on the network.

4.8.2 Transacting on the Network

Nodes on the same network can transact with each other by sending and receiving Ether provided that the sender knows the intended recipient's public address. The second node that joined the network started off with a balance of zero, as shown in Figure 4.15. This


```
> admin.peers
[[{
  caps: ["eth/63", "eth/64", "eth/65"],
  enode: "enode://d7f174f65db61cd883867a49383b5681cec20984da1e645db41ea8cb9f2742d51c127b036b5300fa22cac42441a8fd9959f900a3d9958c11ef4674d78ff6c2a1c@146.231.88.74:52055",
  id: "79735c4f63c9062590a0c42a7aaa037c3992e52b7022012073f8c0480e978f39",
  name: "Geth/v1.9.15-stable-0f77f34b/windows-amd64/go1.14.2",
  network: {
    inbound: true,
    localAddress: "146.231.88.74:3011",
    remoteAddress: "146.231.88.74:52055",
    static: false,
    trusted: false
  },
  protocols: {
    eth: {
      difficulty: 1,
      head: "0x5c30116ae1f79cf0e981cf0bacaaf649b2ad1c9900a8a3a5311ff26ae5bbdb10",
      version: 65
    }
  }
}]
>
```

Figure 4.14: First node checking for other nodes on the network

is because the first node is the only account that was specified to be pre-funded on the genesis file, but the first node can send Ether to the second node.

```
> eth.getBalance("0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69")
0
```

Figure 4.15: Account balance of the second node

To transact on the Geth console, the web3 API is used. It contains many Ethereum libraries that can be used on external applications such as the Geth console application. The sender needs to unlock their account first before they can carry out any transactions on the blockchain, and they need both their public address and the recipient's public address. Figure 4.16 shows a transaction to send Ether from the account in the first node to the account in the second node.

```
> web3.eth.sendTransaction({from:"0xe91d849375e8ba9c0b83b2c8006dd4ec3ca71a29",to:"0xed5d358f2be4ae971eca8e2da7dc5677d177aa69",
value:web3.toWei(300,"ether")})
"0xc03495e4a04782d3a6aa5f4d1312f665f9d5495f0b80f1117b1b56a4f7ebc36"
>
>
```

Figure 4.16: Transaction to send Ether from the first node to the second node

The transaction returned a transaction hash which is a hexadecimal value that uniquely identifies the transaction. The transaction hash is not proof that the transaction has been committed to the blockchain. It is proof that the transaction has been put in a block, but it is not added to the blockchain until the block that it is in has been validated and added to the blockchain.

```
> clique.propose("0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01", true)
null
> clique.proposals
{
  0x86ce3b9540ed1727d55336baeab70aa8d5d02a01: true
}
>
```

Figure 4.18: Command to propose a new sealer node

a sealer node, all the blocks that it validates from that point onwards are rejected and given to other sealer nodes to validate, and it immediately loses its voting rights.

```
> clique.propose("0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01", false)
null
> clique.proposals
{
  0x86ce3b9540ed1727d55336baeab70aa8d5d02a01: false
}
>
```

Figure 4.19: Command to vote out a sealer node

A sealer node can also reverse its proposition to add or remove a sealer node if it has not taken effect yet by being proposed by more than half of the sealer nodes. Figure 4.20 shows how a sealer node can reverse a pending proposal, and running the same command to list the pending proposals returns an empty list as the proposal has been discarded. It is advisable for sealer nodes to discard their proposals if a vote has not been successful after a certain amount of time so that there no hanging proposals on the blockchain.

```
> clique.discard("0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01")
null
> clique.proposals
{}
>
```

Figure 4.20: Command to cancel a proposal that has already been made

4.10 Block Validation on a Clique-based Ethereum Network

When transactions in a blockchain that uses the Clique consensus engine are grouped into a block, the consensus engine will choose the sealer node that should validate the block. To avoid a sealer node from validating a large number of blocks in a row, Clique

uses the formula $N/2+1$, where N is the number of sealer nodes on the blockchain, to determine the number of rounds that a sealer node should wait after validating a block. This formula is meant to prevent sealer nodes from validating successive blocks by making them ineligible to validate a certain number of blocks after successfully validating one.

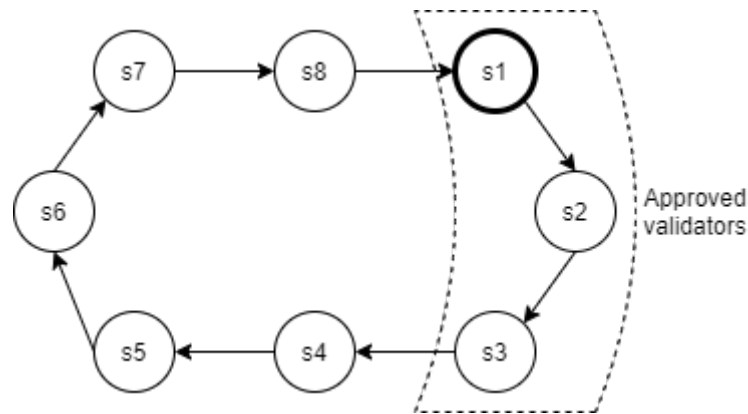


Figure 4.21: Group of sealer nodes at time t_1 where $N=8$

In Clique, a group of multiple nodes can be selected from the pool of sealer nodes to validate a block, but there is one sealer node that is chosen to be the leader of that group of sealer nodes. For any proposed block, there are $N-(N/2+1)$ nodes that are chosen to validate it. The leader of the group of sealers chosen to validate the block has no time delay while the other nodes in the group delay their validation by a random time which gives the leader an advantage to publish the block to the other nodes before the other sealer nodes. Figure 4.21 shows an example of a blockchain with eight sealer nodes at time t_1 . This means that for any block that needs to be validated, there will be a group of three sealer nodes allowed to do the validation according to the formula. In that group of sealer nodes, only one of them will be the leader of the group, which in this example is s_1 . The other two nodes in the group, s_2 , and s_3 , will have a random time delay to make sure that s_1 is the one that will do the validation. If the leader of the group fails to validate the block by the time the first of the two time delays expires, the node with the shorter time delay between s_2 and s_3 will validate the block and publish it to the other nodes in the blockchain. If a sealer node validates a block, they have to wait for five rounds before they can be included in a group of nodes to validate a block.

Figure 4.22 shows the same group of sealer nodes at time t_2 . The assumption is that

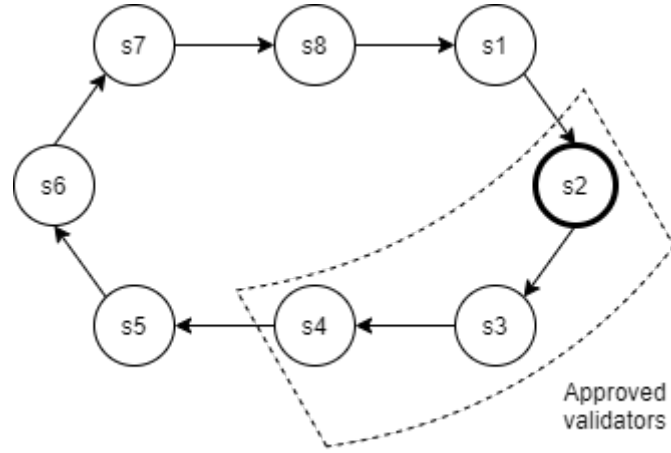


Figure 4.22: Group of sealer nodes at time t_2

node s_1 validated the previous block at time t_1 , which makes it ineligible for the next $N/2+1$ blocks, which in this case is five blocks because $N=8$. When a new block needs to be published, a new group of nodes is chosen by the consensus engine to do the validation. The group will also have three nodes since the value of N is still the same, and this time node s_2 is the leader of the group. Nodes s_3 and s_4 will each have a random time delay before they can publish the block, and so if s_2 is available, it will publish the block before them. The random time delay for the other two nodes is to prevent a situation whereby two or more sealer nodes validate a block at approximately the same time which can create a conflict as some nodes on the blockchain will receive a block published by one sealer node while the other nodes receive a block published by another sealer node.

4.11 Conclusion

In this chapter, some background concepts on Ethereum such as Ethereum clients and the EVM were defined. The primary aim of this chapter of creating a private blockchain network using Ethereum was met. The decision of the appropriate consensus mechanism to use was also made and PoA was chosen due to reasons outlined in the chapter. The blockchain network was tested by initiating a transaction and the theory behind the validation of the transaction was also explained. This blockchain network forms the basis for the peer-to-peer energy trading system as it is where the smart contract will be deployed and it also acts as a medium of payment within the system. The next step is

to design the smart contract and deploy it on the private blockchain network.

Chapter 5

Smart Contract Implementation

This chapter looks at how smart contracts are executed by the Ethereum protocol and then looks at the development of the smart contract for the peer-to-peer energy trading system. The functions of the smart contract will also be tested before the smart contract is deployed to the blockchain network.

5.1 Ethereum Smart Contract Execution

Ethereum smart contracts are executed on the EVM, which is a Turing-complete machine whose state changes when a smart contract is executed [100]. When a smart contract is deployed to a blockchain network, the code of the smart contract is visible to every participant on the network. This code, however, cannot be changed by anyone, including the creator, due to the immutability property of blockchain. When a smart contract is deployed to the blockchain, it is converted to bytecode, which is a set of instructions to the EVM [110]. When the smart contract is called, the bytecode is executed by the EVM, and this changes the state of the EVM. Turing-complete machines allow programs to run infinitely, and this has the potential to cripple the network. To counter this, Ethereum attaches a cost to every transaction that uses the EVM. The cost of the transaction is directly proportional to the amount of computation required by the transaction [111]. This means that executing a smart contract to add two numbers costs less than executing a smart contract to do more complex calculations.

5.2 Smart Contract for the Proposed System

The smart contract is the main program logic for the peer-to-peer energy trading system. The smart contract is designed using the Solidity programming language, which is the native programming language supported by Ethereum [112]. Its purpose in the system can be broken down into several categories which represent different functions on the smart contract code.

5.2.1 Data Storage

The smart contract acts as one of the data storage locations for the system. It stores the list of sellers that are currently selling energy as well as all the details about these listings, such as the price. The function to add a new listing is called `listElectricity`, and it takes two arguments, as shown in Figure 5.1. The first argument is the quantity of electricity that the seller intends to sell, and the second argument is the price that they want to set for that quantity of electricity. The function also assigns an incremental unique identifier call `listCount`. This field uniquely identifies a listing on the smart contract.

```
function listElectricity(string memory _quantity, uint _price) public {
    require(bytes(_quantity).length > 0);
    require(_price > 0);
    listCount++;
    elec[listCount] = Electricity(listCount, _quantity, _price, msg.sender, false);
    emit ListingAdded(listCount, _quantity, _price, msg.sender, false);
}
```

Figure 5.1: Smart contract code for the `listElectricity` function

The `listElectricity` function first validates the user input by checking that the quantity of electricity is a positive integer and that the price is greater than zero. The next step is to create the array of the listing with all its parameters, like the address of the seller and its unique identifier. It is this array that is added to the smart contract by using the `emit` keyword to trigger the `ListingAdded` event that is shown in Figure 5.2.

When the `ListingAdded` event is triggered, this is treated as a transaction as it changes the state of the smart contract. The state of the smart contract reflects the data that the smart contract holds at any point in time. The state of the smart contract is part of

```
event ListingAdded(  
    uint id,  
    uint quantity,  
    uint price,  
    address payable owner,  
    bool purchased  
);
```

Figure 5.2: ListingAdded event code of the smart contract

the state of the EVM, and so any time the state of the smart contract is changed, this changes the state of the EVM, and so there is a cost attached to it. The state of the EVM is visible to members of the blockchain, and so by extension, the state of a smart contract can be viewed by members of the blockchain. This means that the data about the list of sellers is stored in the state of the smart contract, and so it can be changed even after it is added to the smart contract, unlike the smart contract code itself that cannot be changed after it has been deployed to the blockchain.

5.2.2 Transaction Medium

The smart contract is also responsible for handling the payments between the buyer and the seller. It transfers Ether from the buyer's account to the seller's account. When a user intends to purchase electricity from a particular seller, the smart contract first retrieves the details of that listing. This is done by calling the `purchaseElectricity` function as shown in Figure 5.3. A listing is identified using its unique id, and all its details, such as the seller's address, are retrieved by using the listing's ID as an argument for the function. The function also carries out some checks to see if the listing contains valid information such as a valid ID and that the electricity on that listing has not already been purchased by someone else.

Another purpose of this function is to mark the digital asset, which in this case is the electricity, as sold so that no other buyer attempts to purchase it. It also changes the owner property from the seller to the buyer to indicate the transfer. Just like other functions in solidity, the `purchaseElectricity` function triggers an event that changes the state of the EVM, and so it has a cost attached to it. The `purchaseElectricity` function does

```
function purchaseElectricity(uint _id) public {
    Electricity memory _electricity = elec[_id];
    require(_electricity.id > 0 && _electricity.id <= listCount);
    require(msg.value >= _electricity.price);
    require(!_electricity.purchased);
    require(_seller != msg.sender);
    _electricity.owner = msg.sender;
    _electricity.purchased = true;
    elec[_id] = _electricity;
    emit ElectricityPurchased(listCount, true);
}
```

Figure 5.3: purchaseElectricity function of the smart contract

not handle the payment of the seller after a transaction. There are two functions for this purpose. The first one deals with a successful transaction, and the other function handles unsuccessful transactions. The second function is necessary as the physical transfer of electricity from the seller to the buyer can take some time, and there is a possibility of the transfer being interrupted. The smart contract takes this possibility into account by having a function to handle such scenarios.

1. Successful Transaction

The function to handle the payment of the seller by the buyer after a successful transaction is triggered after the total amount of energy listed by the seller has been transferred to the buyer. The smart contract code for this function is shown in Figure 5.4.

```
function paySeller(uint _id) public payable {
    Electricity memory _electricity = elec[_id];
    address payable _seller = _electricity.owner;
    address(_seller).transfer(msg.value);
    emit SellerPaid(listCount, _electricity.quantity, _electricity.price, msg.sender, true);
}
```

Figure 5.4: Smart contract code for the paySeller function

The paySeller function accepts a single argument in the form of the ID of the listing. The first thing that the function does is to retrieve the details of the listing and gets the address of the seller, as this is the address that the Ether should be transferred to. The function also gets the amount that is due to the seller, which in this case

is the amount that was set by the seller when they were listing the electricity for sale. The entire amount is then transferred to the seller's address as the transfer of electricity has been completed.

2. Unsuccessful or Incomplete Transaction

During the transfer of electricity from a seller's battery to the buyer's battery, it is possible for the connection between their batteries to be lost. This can be due to either deliberate action by one of the parties or due to other reasons. The smart contract has to take this into account to ensure that neither of the parties suffers a loss due to this. This is done through the `purchaseFailed` function of the smart contract, whose code is shown in figure 5.5.

```
function purchaseFailed(uint _id, uint _amount) public payable {
    Electricity memory _electricity = elec[_id];
    address payable _seller = _electricity.owner;
    address(_seller).transfer(_amount);
    emit FailedPurchase(listCount, _electricity.quantity, _amount, msg.sender, true);
}
```

Figure 5.5: The `purchaseFailed` function code in the smart contract

This function is similar to the `paySeller` function except for one key difference. It accepts a second argument which is the amount. The `paySeller` function takes the amount stated by the seller when they are entering the details of their listing and pays the seller this amount as they have transferred all the electricity that they were supposed to transfer. However, the `purchaseFailed` function cannot take this amount as it is only called when the transfer of electricity was not completed. This means that only a fraction of the electricity that should have been transferred from the seller to the buyer has been transferred, and so the seller is only entitled to a fraction of the amount they were supposed to receive if all the electricity was transferred. The smart contract is not responsible for calculating how much electricity had been transferred when the process was interrupted or for calculating how much the buyer owes to the seller for the electricity that had already been transferred before the interruption. These calculations are left to the web application and the smart meter, which is the device responsible for measuring how much electricity has been

transferred from the seller to the buyer. This is done to keep the computations on the smart contract as low as possible to minimize transaction costs. When the calculations have been made, the `purchaseFailed` function is invoked with the unique ID of the listing as well as the calculated amount that is owed to the seller. The smart contract will then transfer the stated amount from the buyer's account into the seller's.

5.3 Alternative Approach to the Smart Contract Design

An alternative approach to designing the smart contract would have been to use the smart contract as an intermediary storage location during the physical transfer of electricity from the seller to the buyer. When a buyer purchases electricity from the seller, the total amount of Ether would be transferred from the buyer's account and held in the smart contract while the physical transfer of electricity is taking place. Once this transfer is complete, the smart contract would then transfer the Ether that it is holding to the seller's account. If the transfer of electricity is interrupted, the smart contract would transfer the amount that is owed to the seller for the electricity that had been transferred prior to the interruption and then refund the buyer the rest of their Ether.

The major flaw with this approach is that it is costly due to the number of transactions that need to be done for a single purchase. Whenever a buyer attempts to buy electricity, the Ether will be transferred to the smart contract, and this transaction has a transaction cost attached to it, and there will be another cost when the smart contract has to then transfer the Ether to the seller upon the successful completion of the transfer of electricity. For an interrupted transaction, there are three transactions that each cost Ether. There is the transaction of transferring Ether from the buyer's account to the smart contract, and then the smart contract has to initiate two transactions. The first transaction is to pay the seller for the electricity supplied, and the other transaction is to refund the buyer the rest of the Ether they would have transferred to the smart contract. This is in comparison to the single transaction for both successful and unsuccessful transactions

required by the proposed approach. The Ether is not transferred from the buyer's account until the conclusion of the transfer of electricity.

5.4 Smart Contract Functionality Testing

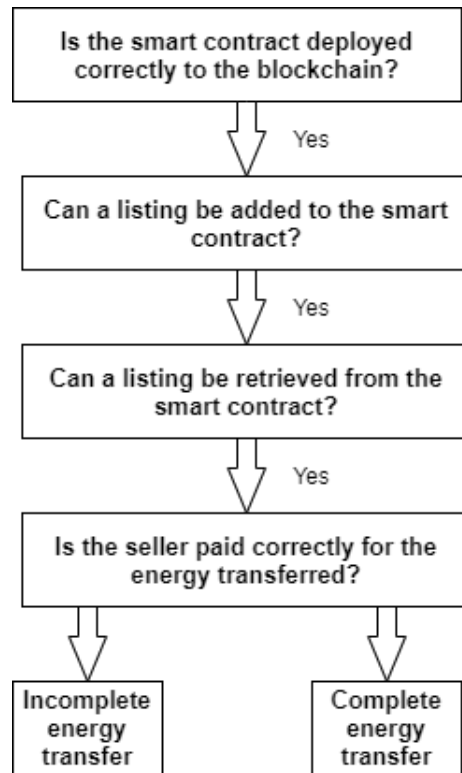


Figure 5.6: Smart contract test questions

The purpose of these tests is to check that each function in the smart contract performs as expected. The test questions that need to be answered by the tests are shown in Figure 5.6. The tests are done by writing a test program that gives the smart contract known data, with a known expected outcome. If the smart contract produces the expected results, then the test is successful, but if the results deviate from the expected results, then that means there is an error in the smart contract code. To carry out the test, the smart contract is deployed to a Ganache test network. Ganache is a software that simulates an Ethereum network, but it is not a blockchain network itself [113]. Truffle is used to run the test scripts as it has the necessary tools for this purpose.

The first test was to check if the smart contract deploys correctly to the blockchain and that it has a name. The test also checked that the `listElectricity` function worked as

expected. The test program simulated the creation of a new listing to see if the function successfully created a new listing. The JavaScript code for the test to check for the successful deployment of the smart contract is shown in Figure 5.7.

```
describe('deployment', async() => {
  it('deploys successfully', async() => {
    const address = await marketplace.address
    assert.notEqual(address, 0x0)
    assert.notEqual(address, '')
    assert.notEqual(address, null)
    assert.notEqual(address, undefined)
  })

  it('has a name', async () =>{
    const name = await marketplace.name()
    assert.equal(name, 'Peer to Peer Energy Trading')
  })
})
```

Figure 5.7: Test code to check smart contract deployment

The test method first checked if the smart contract was deployed properly to the blockchain. This was done by checking if the smart contract has a valid address on the blockchain to which it was deployed. The program retrieved the address of the smart contract and checked if it was not null. The next step was to check that the smart contract was deployed with the correct name. This test was done by retrieving the name of the smart contract from the blockchain and comparing it to a string of the correct name. If these values match, then the smart contract has the correct name. Figure 5.8 shows the results after running the test program. The test was successful as the smart contract was deployed correctly to the blockchain, and the name matched as well.

```
Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Contract: Marketplace
  deployment
    ✓ deploys successfully
    ✓ has a name
```

Figure 5.8: Results after running test program

The next test was to check if the `listElectricity` function works correctly. The test function for this operation is shown in Figure 5.9.

```
describe('electricity', async() => {
  let result, listCount

  before(async () => {
    result = await marketplace.listElectricity('200', web3.utils.toWei('1', 'Ether'))
    productCount = await marketplace.listCount()
  })

  it('list Electricity', async () =>{
    assert.equal(listCount, 1)
    console.log(result.logs)
  })
})
```

Figure 5.9: Test code to retrieve electricity listings

The test function tested if the smart contract could add a new listing, and it did this by calling the `listElectricity` function of the smart contract. This function accepts two arguments, the quantity, and the price. These two arguments were passed as strings by the test program. The last part of the test function is to print the results of the operation, and so if the listing is successfully added, it should be printed on the console. The results of running the test program are shown in Figure 5.10.

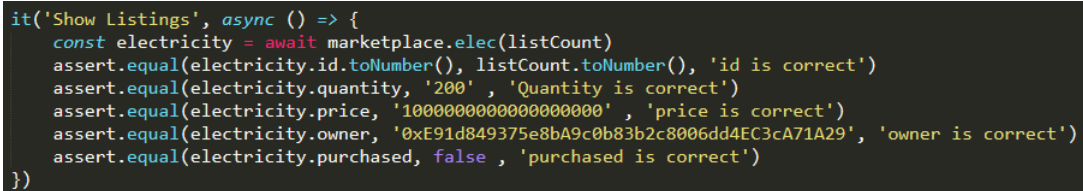
```
electricity
[
  {
    address: '0x1eA9d1Cce756b489D7f47Cb32659FA89aedF7260',
    blockNumber: 11086,
    transactionHash: '0x1671723e9a87c1273a57f816dd6bb77a928d8fad24caa0aa6b64d34a062ca77b',
    transactionIndex: 0,
    blockHash: '0x59fbdc88f2c009416967a6f11231ec664b0002d2c14f3991b39350943317c900',
    logIndex: 0,
    removed: false,
    id: 'log_5f631b54',
    event: 'ProductCreated',
    args: Result {
      '0': [BN],
      '1': '200',
      '2': [BN],
      '3': '0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29',
      '4': false,
      __length__: 5,
    },
  },
]
```

Figure 5.10: Results after running test program to retrieve listings

The test function returned an array with the details of the listing. The array contained data such as the block number of the block that contains this contract call transaction. The array also had the transaction hash and the hash of the block. The “Result” sub-block contained the details of the listing that was added. It had the quantity of 200 that

was specified in the test function and the address of the node that made the contract call. The same test was done, but the quantity argument was left empty. The test returned an error message, and the listing was not added to the smart contract. This is because the listElectricity function first checks that the function call has the required number of arguments and that these arguments are not equal to or less than zero. This shows that this test was successful.

The next test was to check if the listing that had been added to the smart contract could be retrieved and that it contained the information that was added in the previous test. Figure 5.11 shows the code to perform this test.



```
it('Show Listings', async () => {
  const electricity = await marketplace.elec(listCount)
  assert.equal(electricity.id.toNumber(), listCount.toNumber(), 'id is correct')
  assert.equal(electricity.quantity, '200', 'Quantity is correct')
  assert.equal(electricity.price, '1000000000000000000', 'price is correct')
  assert.equal(electricity.owner, '0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29', 'owner is correct')
  assert.equal(electricity.purchased, false, 'purchased is correct')
})
```

Figure 5.11: Test code to check the accuracy of the data retrieved

The listing was first retrieved from the smart contract, and then each of its parameters was compared to the known data that was entered in the previous test. The ID, quantity, price, owner's address, and whether or not the listing has been purchased were all checked. The test only passes if all the comparisons return true. In this case, all the details of the listing were correct, which means that the data was entered correctly on the smart contract and could be successfully retrieved.

The last functionality test for the smart contract was to see if the seller was successfully paid after a transaction. The test function for this purpose is shown in figure 5.12.

The first step was to get the account balance of the seller's address before the transaction. The web3 library was used for this purpose. After getting the balance, the paySeller function of the smart contract was called. This function only takes one argument, which is the ID of the listing that is to be sold, but a second argument can be passed to it. The second argument contains the details of a transaction to transfer Ether from one account to another. In this case, 1 Ether was to be transferred from the buyer's account. The address of the buyer was defined outside this test function as it is used in multiple

```
it('Sells electricity', async () => {
  let oldSellerBalance
  oldSellerBalance = await web3.eth.getBalance('0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29')
  oldSellerBalance = new web3.utils.BN(oldSellerBalance)

  result = await marketplace.paySeller(listCount, { from: buyer, value: web3.utils.toWei('1', 'Ether')})

  const event = result.logs[0].args
  assert.equal(event.id.toNumber(), listCount.toNumber(), 'id is correct')
  assert.equal(event.quantity, '200', 'quantity is correct')
  assert.equal(event.price, '1000000000000000000', 'price is correct')
  assert.equal(event.owner, buyer, 'owner is correct')
  assert.equal(event.purchased, true, 'purchased is correct')

  let newSellerBalance
  newSellerBalance = await web3.eth.getBalance('0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29')
  newSellerBalance = new web3.utils.BN(newSellerBalance)

  let price
  price = web3.utils.toWei('1', 'Ether')
  price = new web3.utils.BN(price)

  const expectedBalance = oldSellerBalance.add(price)

  assert.equal(newSellerBalance.toString(), expectedBalance.toString())
})
```

Figure 5.12: Test code to check if the seller is paid after a transaction

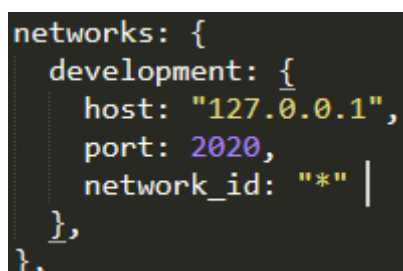
functions within the test program. The next step was to check the output of the transaction and compare it with known values to see if the transaction was successful. The “purchased” parameter of the listing had changed from false to true to indicate that the listing had been purchased. The next section of the test function retrieved the account balance of the seller’s account after the transaction. This new account balance was then compared to an expected balance which is the sum of the old balance and the price of the electricity that the seller just sold. If these two balances match, this means that the seller has received the Ether that was due to them, and the `paySeller` function of the smart contract works as it should.

The test function for the `purchaseFailed` function of the smart contract was similar to the test function for the `paySeller` function. The only difference was in the value that was passed in the argument when calling the function. In the test function for an incomplete transaction, the application called the `purchaseFailed` function with the ID of the listing as the first argument and the details of a transaction to send Ether from the buyer’s account. The value of the transaction was not 1 Ether as the transfer of electricity was not completed. The value to be passed is instead received from the web application, and to test the function, a value of 0.5 Ether was used. The result of the test after running it through the Truffle test application, was the same as for the previous test as both tests

were for the transfer of Ether from one account to another.

5.5 Deploying the Smart Contract

After the smart contract had been successfully tested, the next step was to deploy it to the private blockchain network. To deploy the smart contract, the Truffle application was used. A configuration file that contains details about the deployment had to be created first. This file is used by Truffle to identify the blockchain network to deploy the smart contract to. Figure 5.13 shows part of this configuration file. The code snippet contains network information about the deployment. The `network_id` parameter is set to accept all networks, but it can also be set to the specific network ID of the network where the smart contract is to be deployed. This is not a problem, however, as the `port` parameter is used to identify the network. Only a single blockchain network can use a port at any given time, so Truffle will use the port to identify the network. The port that the blockchain network should use was set when starting up the Geth console.



```
networks: {
  development: {
    host: "127.0.0.1",
    port: 2020,
    network_id: "*" |
  },
},
```

Figure 5.13: Snippet of the Truffle configuration file showing network settings

After setting these parameters, the smart contract could now be deployed to the network using Truffle. Figure 5.14 shows the results after deploying the Marketplace smart contract to the blockchain. The results contain some information about the deployment, such as the account that deployed the smart contract to the blockchain, the smart contract's address on the blockchain, and the time it took for the smart contract to be deployed. There is also some information relating to the cost of the deployment, such as the gas price and the total cost of the transaction. After the smart contract was deployed to the blockchain, the next step was to design the client-side application.

```

2_deploy_contracts.js
=====

Deploying 'Marketplace'
-----
> transaction hash: 0xa54461b66b70625ad9f41c5596b590f006bc2ad7b503f6736d6abb55f4b16dbd
> Blocks: 0 Seconds: 12
> contract address: 0x9bac688636F47aC86b9cb440eCf3e542369f57Be
> block number: 1426
> block timestamp: 1615304275
> account: 0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29
> balance: 4999999700
> gas used: 737545
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.0147509 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.0147509 ETH

```

Figure 5.14: Results after successfully deploying the smart contract to the blockchain

5.6 Client-side Application

This section looks at the design and implementation of the client-side application for the peer-to-peer energy trading system as well as other tools that enable users to use the application. The client-side application serves two purposes in the application. The main purpose is to call the functions in the smart contract according to the user's commands as well as interpret the data from the smart meter and give the user accurate feedback. The secondary purpose of the client-side application is to provide the user with a user-friendly interface that makes it easy to navigate the system.

5.6.1 Client-side Application and Smart Contract Interaction

This section looks at how the client-side application calls functions in the smart contract and the technologies that are used to achieve this interaction. Both the backend and the frontend for the client-side application were developed using React JS programming language. React JS is a JavaScript library for building user interfaces, and it is used to fetch data from both the smart contract and the smart meter and display it to the user. The first step in building the backend of the application is to create the functions to call the smart contract functions. The functions contained in the backend of the application have the same names as those in the smart contract as each of these functions calls one

function in the smart contract. These functions also have the same number of arguments as the ones in the smart contract, and they pass these arguments to the smart contract. Figure 5.15 shows the code for one of the functions contained in the client-side application. This function calls the `listElectricity` function in the smart contract, and it passes two

```
listElectricity(quantity, price) {  
  this.setState({ loading: true })  
  this.state.smartContract.methods.listElectricity(quantity, price).send({ from: this.state.account })  
  .once('receipt', (receipt) => {  
    this.setState({ loading: false })  
  })  
}
```

Figure 5.15: Code for the `listElectricity` function of the client-side application

arguments which are the quantity and the price. The function also passes an account responsible for the transaction costs. This is because the process of adding a new listing requires a smart contract call, and this has a transaction cost attached to it. The function gets a receipt after calling the smart contract. This receipt from the smart contract is either a notification of the success of the contract call or an error message if the process was not successfully completed. The `setState` function is an inbuilt React JS function that tells the browser to re-render the page because of a change that has been made. This is necessary so that the web page automatically refreshes after a new listing is added.

5.6.2 Client-side Application Frontend

The purpose of the frontend of the web application is to provide the user with a user-friendly interface that allows them to interact with the system. This means that the frontend of the application should support all the major functions of the system, such as allowing a seller to list electricity for sale and for a potential buyer to view all the listings and be able to purchase electricity. The application should also be able to facilitate the payment for the transaction between the buyer and the seller. To do this, Metamask is used.

Metamask

Metamask is one of the smart contract tools that were specified in the software architecture. It is a browser extension that allows interaction between users and Ethereum blockchain networks through a web browser. Metamask comes preconfigured with the

Ethereum Mainnet as well as the other popular test networks, but it also allows users to add other networks. Figure 5.16 illustrates how users can add the private blockchain network on Metamask. The user just needs to know the RPC port that the blockchain network is running on and the network ID of the blockchain network. The other thing that the user needs to add to their Metamask is their Ethereum account. Metamask allows users to upload their encrypted private key, which is unlocked when the user enters their password that was set when the account was created.

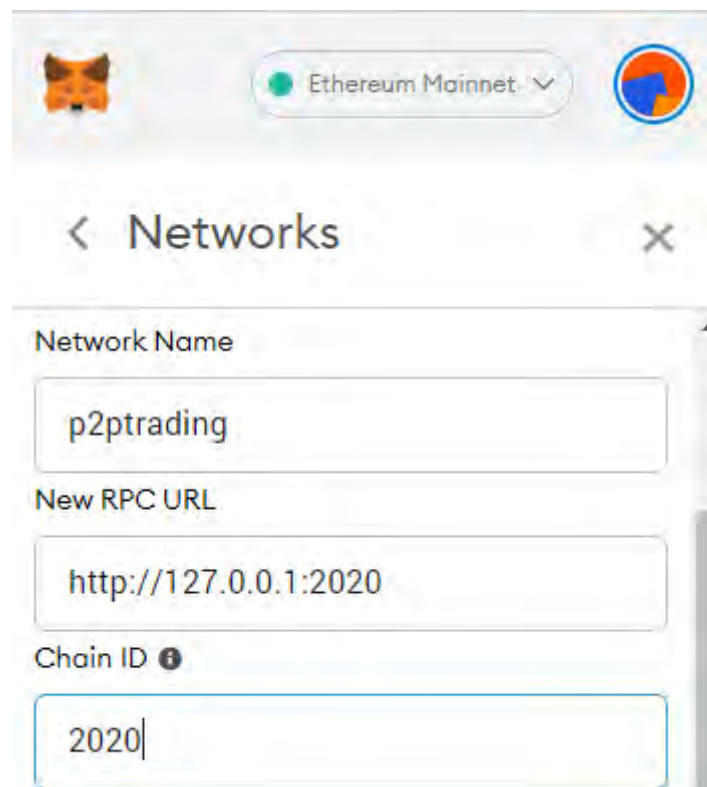


Figure 5.16: Screenshot showing the parameters required to add a network to Metamask

Every user in the system needs to have Metamask set up on their browser if they intend to use the peer-to-peer energy trading system. This is because some of the data that is displayed on the web application has to be retrieved from the blockchain, and access to it is only restricted to members of the blockchain. A function to check that the browser has support for Ethereum distributed applications is included in the client-side application. Some browsers have native support for Ethereum applications, but some like Chrome require a third-party application such as Metamask in order to be able to interact with Ethereum networks. The code for the function is shown in Figure 5.17.

This function is called when a user tries to access the web application. Metamask then checks if the account that was added to it is a part of the blockchain network that the smart contract was deployed to.

```
async loadWeb3() {
  if (window.ethereum) {
    window.web3 = new Web3(window.ethereum);
    await window.ethereum.enable();
  }
  else if (window.web3) {
    window.web3 = new Web3(window.web3.currentProvider);
  }
  else {
    window.alert('Your browser is not supported. You should consider trying MetaMask!');
  }
}
```

Figure 5.17: Code for checking if the browser supports Ethereum distributed applications

The function first checks if the browser has support for the versions of Ethereum that are still supported, and if it does, then the application loads as it should. If the browser does not have the newest versions of Ethereum, the function then checks if the browser has support for the legacy versions of Ethereum. If it does, then the function calls the appropriate function from the legacy version of Ethereum to allow the application to load correctly. If both these checks fail, the function returns an alert window informing the user that their browser is not supported and, therefore, the application will not be able to run on that browser

User Interface

The user interface of the client-side application consists of a single page where sellers list electricity for sale, and potential buyers are able to view these listings as well as purchase electricity. Figure 5.18 shows the user interface of the system with one listing already added. The address of the Ethereum account that the user logged into Metamask with is retrieved and displayed on the navigation bar.

The application is divided into three sections. The first section is where a user intending to sell electricity adds a new listing. This section has two text boxes where the user enters the amount of electricity that they intend to sell and the price of the electricity. The button calls the `listElectricity` function of the client-side application, which in turn calls the smart contract function of the same name. The values entered in the text boxes

Peer to Peer Energy Trading

0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29

Add New Listing

Electricity Quantity (mAh)

Electricity Price (Eth)

Add Listing

Buy Electricity

Listing ID	Quantity	Price	Seller	
4	200 mAh	2 Eth	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	<div>BuyRemove</div>

Sellers' Electricity Balances

Seller	Current Balance	Last Updated
--------	-----------------	--------------

Figure 5.18: Screenshot of user interface of the web application

are passed as the arguments along with the Ethereum address that appears at the top of the web page.

The next section of the web page contains a list with the details of each listing. Each listing has an ID, the quantity, the price, and the address of the seller. Each listing also has two buttons. The first button is the Buy button, and this is where a potential buyer clicks if they want to purchase electricity. When clicked, this button calls the `purchaseElectricity` function of the client-side application. This function will then do two actions. It will communicate with the seller's microcontroller to transfer electricity to the buyer, and it will also call the `purchaseElectricity` function of the smart contract. The Remove button removes the listing from the client-side application. This is done by changing the purchased status of the listing to true. The client-side application will first verify that the address of the seller that is retrieved from the blockchain matches with the address that is retrieved from Metamask. If the addresses match, then this will confirm that the user is the owner of the listing; otherwise, they will receive an error message.

The last section of the web page is used to display the electricity balances of all the sellers that have a listing in the system. This is to aid all potential buyers to verify

that the seller they are buying from has enough electricity that they intend to sell. This section is populated with data from the smart meter.

5.7 Client-side Application Access Control

Access to the client-side application is restricted, and only participants in the private blockchain network should be able to access the application. To enforce access control, Metamask is used. Metamask first verifies that a user has the private key to the account that they add and that they have the correct password to decrypt this private key. Metamask will then check that the account is a node in the network that is running on Metamask, and it also retrieves the account balance of the node in the network. The client-side application leverages these features of Metamask to eliminate the need for extra access control mechanisms on the application. The web application first retrieves the details of the account from Metamask and checks if it is connected to the network running on Metamask. The application will only proceed if the account is connected to the account running on Metamask. The application will then check that the network that is running on Metamask is the same network as the one where the smart contract was deployed. This also has to be true for the application to load the data from the blockchain. This means that if a user tries to cheat the system by connecting to the Mainnet, for example, the second condition will not be met as the smart contract is not deployed to that network, and so the application will not load. This security feature will be tested fully during the testing phase.

5.8 Conclusion

The smart contract has been designed and tested and then deployed to the private blockchain network after passing all the test scenarios. The client-side application has also been developed and integrated with the smart contract. The next step is to configure the hardware components and integrate it with the smart contract and the client-side application.

Chapter 6

Hardware Implementation

In this chapter, the final module of the peer-to-peer energy trading system which consists of the hardware components is introduced. In this chapter, the hardware components will be identified, configured and integrated into the proposed system.

6.1 Hardware Components Considerations

In this section, the technical requirements for each hardware component will be specified, and a choice will be made for the exact hardware components that will be used in the peer-to-peer energy trading system.

6.1.1 Smart Meter

The purpose of the smart meter in the system has already been outlined, and there are multiple smart meters that can serve this purpose. There are, however, a few technical specifications that the smart meter should meet. The first aspect to take into consideration is the parameters that the smart meter measures. Most smart meters measure the basic electrical parameters such as voltage, current, and power, but some smart meters go above and beyond this and use the figures for these parameters to calculate other useful values like the amount of electricity used over a period of time. Another important consideration is the power requirements of the smart meter. The system to be developed is a small-scale prototype of the real-world system, and so the smart meter needs to be energy-efficient so that it does not consume a huge portion of the energy that is produced in the system.

The smart meter also needs to be able to handle a wide range of voltages so that the system can be scalable without making too many design changes. The last technical requirement is that the smart meter should be customizable to suit different requirements. That means that it should be possible to change the firmware that comes pre-installed on the smart meter in order to increase the number of possible application areas for the smart meter. Some smart meters come installed with firmware that does not support HTTP, for example, but a simple change in the firmware can add this functionality. The Sonoff POW R2 smart meter meets the criteria for the ideal smart meter, and so it is used in the system.

6.1.2 Microcontroller

The microcontroller contains the program logic for the hardware components of the system. The microcontroller receives instructions from the client-side application to open a particular relay which will transfer electricity to the node that that relay is connected to. Therefore the microcontroller has to have internet capabilities in order to receive these instructions. It also needs to have general-purpose input/output (GPIO) pins, as these are used to control the relays. The microcontroller only needs to have digital GPIO pins as they are only used to send on/off signals to the relay as opposed to other operations that require analogue signals.

There are various microcontrollers that are suitable for this purpose, but the Raspberry Pi is the most appropriate choice as it also serves another purpose in the system. The Raspberry Pi is used in the system as the nodes in the blockchain network, and so to reduce the number of hardware components in the system, the Raspberry Pi is also used as the microcontroller for the hardware part of the system.

6.1.3 Battery

The main requirement for the battery is a known or easily measurable capacity. This eliminates the need for a different meter to measure how much power is left in the battery at any given point. Another requirement for the battery is for it to have different input and output ports. This is because, typically, batteries with one set of input and output

terminals cannot transfer energy to one another unless one of them has a higher voltage than the other. This is because current only flows from high potential to low potential, and this means that if two batteries both have 12 Volts, they cannot charge each other unless one of the two voltages is stepped up using a DC-DC voltage converter [114]. The same also applies when trying to charge a battery with a higher capacity from a battery with a lower capacity.



Figure 6.1: Power bank used in the peer-to-peer energy trading system

To avoid having to use a DC-DC voltage converter on every battery in the system, a power bank is more ideal. A power bank is a portable battery with an electric circuit for controlling the power going in or out of it. It is more suitable for this purpose than a standard battery because it has separate input and output ports. This allows it to charge other batteries regardless of their capacities. This means a power bank can charge a battery with a much larger capacity than its own as well as one with the same or smaller capacity. The output port of a power bank allows it to discharge until it is empty. This means it can power a load like a normal battery until it is empty, and the output ports can also be used to charge other batteries. Figure 6.1 shows an image of one of the power banks used in the system.

6.2 Sonoff POW R2 Smart Meter

The Sonoff POW R2 is a smart meter that monitors home energy usage. It can also be used as a smart switch as it can also stop current from flowing through it. The Sonoff smart meter measures the current, power, and voltage that flows through it. It connects to a Wi-Fi network which allows a user to control it remotely and view the information that the smart meter captures from another device. The Sonoff smart meter only measures the current that flows in a single direction, and so two smart meters are used for each node. The first smart meter records the amount of energy that is being used by the node, and the second smart meter is used to measure the energy that is going into the node's battery from other nodes.

6.2.1 Sonoff Smart Meter Schematic

This section takes an in-depth look the components that make up the Sonoff POW R2 smart meter. Figure 6.2 below shows some of the components that make up the smart meter.

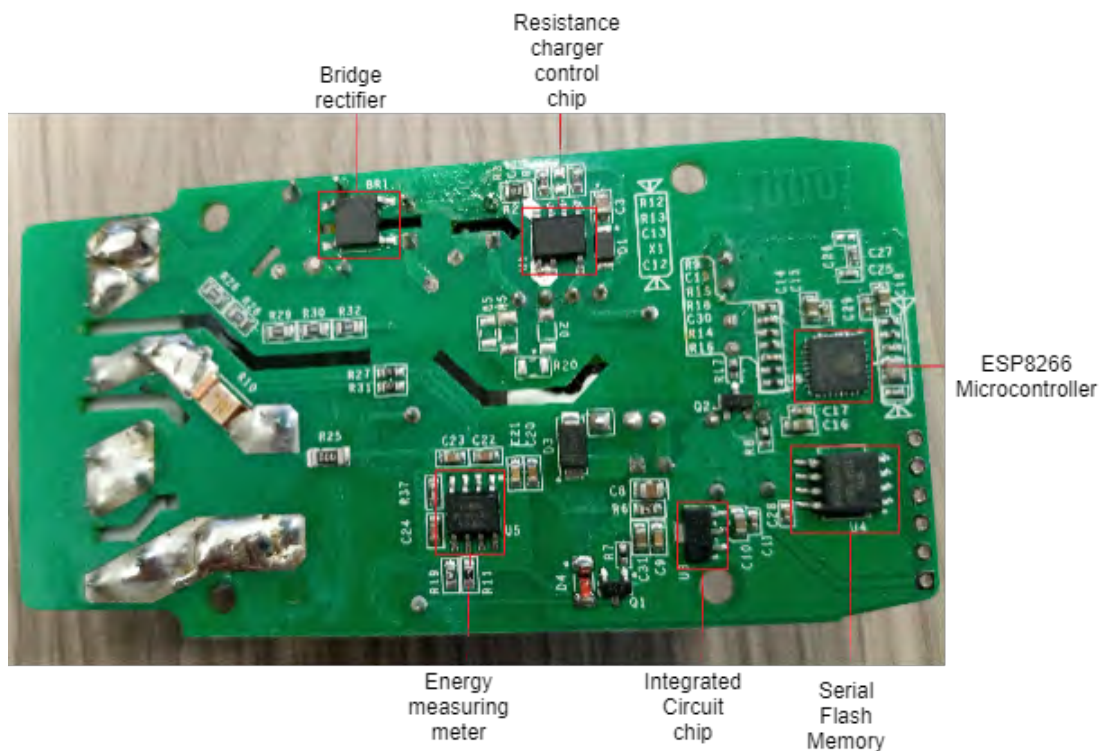


Figure 6.2: Components of Sonoff POW R2 smart meter

1. ESP8266 Microcontroller

The Sonoff POW R2 uses an ESP8266 microcontroller. The ESP8266 contains an 80MHz processor, 32 KB of RAM, and 16 GPIO pins [115]. It also has an inbuilt Wi-Fi microchip which allows it to connect to networks using the IEEE 802.11 b/g/n protocols, and it also comes with a full TCP/IP stack.

2. Flash Memory

The Sonoff smart meter comes with 32M-bit flash memory that holds the firmware for the smart meter. Flash memory is a low-cost non-volatile type of memory that can be erased and reprogrammed electrically [116]. It is an example of electronically-erasable programmable read-only memory (EEPROM). The smart meter also has serial ports, as shown in the figure below. These serial ports allow the smart meter to be connected to a computer where this flash memory can be reprogrammed with custom software.

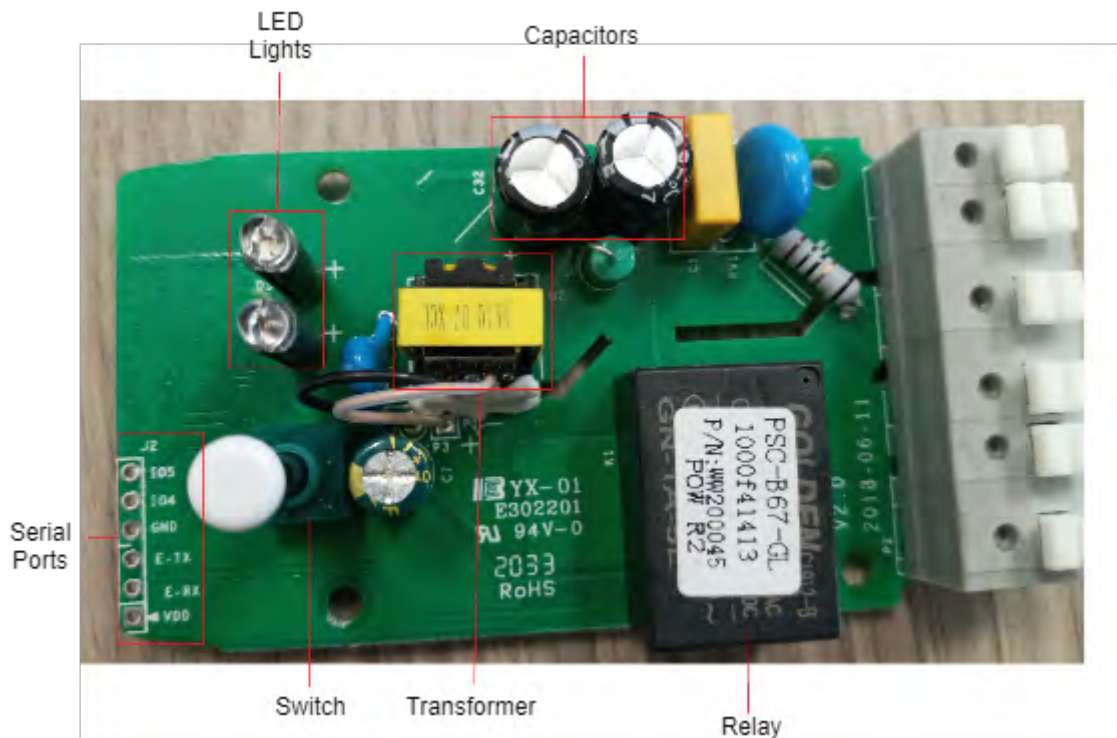


Figure 6.3: Top view of Sonoff POW R2 smart meter

Figure 6.3 above shows the top section of the Sonoff POW R2 smart meter. It contains the serial ports, a switch, and some LED lights that show the state of the smart meter.

The smart meter also has a relay that allows it to operate as a remote switch, and it also contains a step-down transformer. A transformer is an electrical device that transforms electrical power from one voltage and current setting to another. A step-down transformer decreases an incoming current's voltage, and in the Sonoff smart meter, it is used to reduce the voltage to a level that is suitable for the components of the smart meter as a higher voltage may damage them.

6.2.2 Sonoff POW R2 Technical Specifications

The Sonoff POW R2 has an ideal operating range for different parameters as shown in Table 6.1. The smart meter should not be used outside of these parameters as it might damage the smart meter and put the user at risk of exposure to electrical current.

Parameter	Description
Voltage Range	100V – 240V AC
Maximum Current	15A
Maximum Power	3500W
Operating Temperature	0°C to 40°C
Operating Humidity	5% to 90% Non-condensing

Table 6.1: Sonoff POW R2 technical specifications [117]

6.2.3 Sonoff POW R2 Wireless Specifications

Table 6.2 below describes the network standards that the Sonoff smart meter uses.

Parameter	Description
Wireless Frequency	802.11 b/g/n
Security Mechanism	WPA-PSK/WPA2-PSK
Frequency Range	2.4GHz – 2.5GHz
Network Protocols	IPv4, TCP/UDP/HTTP/FTP

Table 6.2: Sonoff POW R2 wireless specifications [117]

6.3 Sonoff POW R2 Firmware

The Sonoff smart meter comes pre-installed with the official Sonoff firmware. The pre-installed firmware has a number of disadvantages, and the main one is that it can only

connect to one application called eWeLink. This application is very restrictive in what it allows the user to do with the smart meter. It shows the user the readings of the meter, but it cannot be integrated with other systems. The application is essentially a remote LED screen that just shows the readings of the smart meter but does not give a user an option to do anything with that information. The firmware also prevents the smart meter from sending out its readings to any other application, and so some of the data transmission protocols that the smart meter contains are used for one-way communication from the smart meter to the application.

For this reason, the smart meter has to be configured with firmware that offers more options, such as querying the smart meter in real-time or storing the readings at a fixed interval to a remote database. The Sonoff smart meter has electronically-erasable flash memory and has serial ports that enable it to be connected to a computer to change the firmware. The Tasmota firmware is the ideal firmware for this device.

6.4 Tasmota Firmware

Tasmota is an open-source firmware made specifically for ESP8266-based devices by Theo Arends [118]. It offers HTTP and MQTT network protocols that enable the devices to be controlled remotely. Tasmota has a web-based interface where the user can view information about energy usage and configure the device. Figure 6.4 below shows the home screen for the Tasmota web application after flashing the firmware onto the Sonoff smart meter.

The first section contains information about power usage. The Sonoff smart meter measures the voltage, current, and power flowing through it. The Tasmota firmware then uses these values to calculate the following parameters.

1. Reactive Power

Reactive power (Q) is the power that continuously bounces back and forth between the source and the load [119]. Reactive power is an important part of voltage control as it can be increased and decreased to keep the voltage going to the load constant and within the accepted voltage range of the load. Equation 6.1 below shows the

formula for calculating reactive power where V is the voltage and I is the current.

$$Q = VI \sin \theta \quad (6.1)$$

2. Apparent Power

Apparent power (S) is a product of the current and the voltage if and only if the phase angle differences between the voltage and the current are ignored [120]. It is also known as the demand, and it is the measure of the amount of power that is used by a load during a particular time. The formula for apparent power is shown in Equation 6.2 where V is the voltage and I is the current.

$$S = VI \quad (6.2)$$

3. Power Factor

Power factor (Pf) is the ratio of working power to apparent power [121]. The formula for the power factor is shown below.

$$P_f = P/S \quad (6.3)$$

The Tasmota firmware also utilizes the memory of the Sonoff smart meter to store information about the total energy used on that particular day and on the previous day. The smart meter also stores the total energy that has been used since the smart meter was powered on. However, all this data is stored in an erasable storage location within the smart meter, and so it is lost when the smart meter is reset. To preserve this data, a more permanent storage solution such as cloud storage has to be used. The data collected by the smart meter is used to determine how much energy is remaining in a node's battery. The battery's maximum storage capacity is stored in the database, and the smart meter updates this figure as the load uses up energy or as energy goes into the battery. The value in this field is calculated by first converting the value in the "Energy Total" field to Ampere-hours (Ah) and then subtracting it from the known capacity of the battery. The values stored in the smart meter can be queried using two protocols, namely MQTT and HTTP, but MQTT is the chosen protocol for this purpose for reasons stated in section 3.7.1.

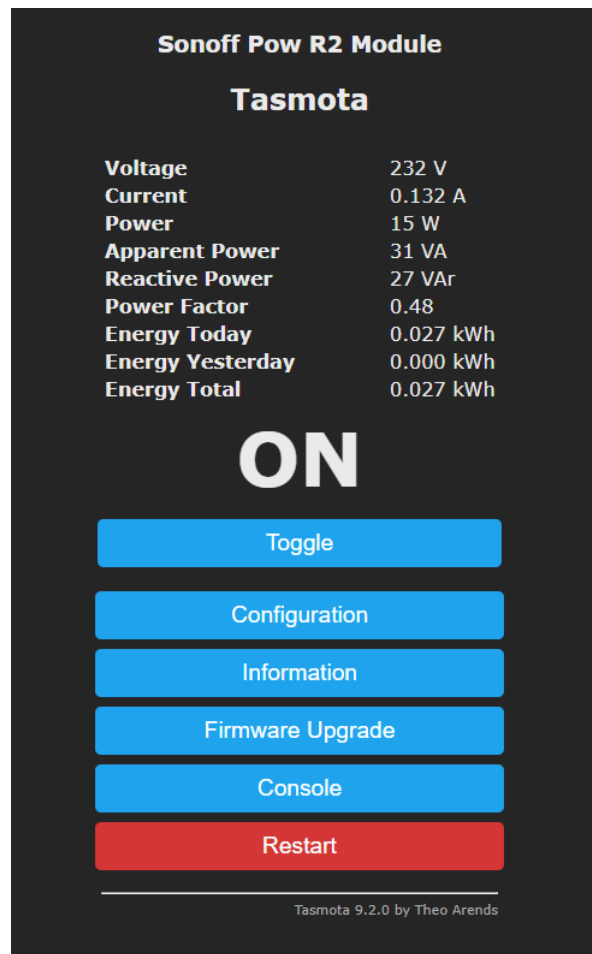


Figure 6.4: Tasmota web interface home page

6.4.1 Tasmota MQTT Configuration

The first thing is to configure the IP address and the port of the MQTT broker. The configuration page on the Tasmota web interface is shown in Figure 6.5. The Sonoff smart meter operates as both an MQTT publisher and a client, which means it can transmit messages to the broker and also receive messages from other publishers. The topic that the smart meter broadcasts to is also set in the MQTT configuration, and in this case, the topic is called `sonoff-pow1`. Any client that subscribes to this topic will be able to receive the messages that are broadcast by the smart meter. To uniquely identify each smart meter in the system, they will all have a different topic to enable a client to know which smart meter broadcast the message.

The smart meter can also subscribe to a topic which allows it to receive commands from elsewhere. This enables the smart meter to be switched on and off remotely through

Sonoff Pow R2 Module

Tasmota

MQTT parameters

Host ()
http://146.231.88.74

Port (1883)
1880

Client (DVES_07BABF)
DVES_%06X

User (DVES_USER)
DVES_USER

Password ☐
....

Topic = %topic% (tasmota_07BABF)
sonoff-pow1

Full Topic (%prefix%/%topic%/)
%topic%/%prefix%/

Save

Configuration

Figure 6.5: Tasmota MQTT configuration

MQTT. The rate at which the smart meter broadcasts data to the MQTT broker is also configured on the Tasmota web interface, as shown in Figure 6.6. The telemetry period can be set to a high-frequency rate due to the low bandwidth usage of the MQTT protocol, and in this case, it has been set to send data every 30 seconds.

Tasmota provides a console log that shows the data that is being transmitted from the smart meter. Figure 6.7 below shows the log of the smart meter on the Tasmota web interface.

The MQTT topic is shown on the log, and in this case, it queries the state of the smart meter and returns a result in JSON format. The same JSON data is transmitted to the MQTT broker, where it is broadcast to the clients. The data that is sent by the smart meter is shown more clearly in array form in Figure 6.8 below.

The MQTT broker that is used is the Mosquitto MQTT broker. It is an open-source message broker that receives messages from publishers and distributes them to the clients

Sonoff Pow R2 Module

Tasmota

Logging parameters

Serial log level (Info)

Web log level (Info)

Mqtt log level (None)

Syslog level (None)

Syslog host ()

Syslog port (514)

Telemetry period (300)

Figure 6.6: Tasmota time period configuration

that have subscribed to that topic.

```

16:33:07 RSL: sonoff-pow1/tele/INFO3 = {"RestartReason":"Software/System restart"}
16:33:07 RSL: sonoff-pow1/stat/RESULT = {"POWER":"ON"}
16:33:07 RSL: sonoff-pow1/stat/POWER = ON
16:33:10 QPC: Reset
16:33:11 RSL: sonoff-pow1/tele/STATE = {"Time":"2021-03-26T16:33:11","Uptime":"0T00:00:09","UptimeSe
16:33:11 RSL: sonoff-pow1/tele/SENSOR = {"Time":"2021-03-26T16:33:11","ENERGY":{"TotalStartTime":"20
16:33:38 LOG: SerialLog 0, WebLog 2, MqttLog 0, SysLog 0, LogHost , LogPort 514, TelePeriod 30
16:33:41 RSL: sonoff-pow1/tele/STATE = {"Time":"2021-03-26T16:33:41","Uptime":"0T00:00:39","UptimeSe
16:33:41 RSL: sonoff-pow1/tele/SENSOR = {"Time":"2021-03-26T16:33:41","ENERGY":{"TotalStartTime":"20
16:34:11 RSL: sonoff-pow1/tele/STATE = {"Time":"2021-03-26T16:34:11","Uptime":"0T00:01:09","UptimeSe
16:34:11 RSL: sonoff-pow1/tele/SENSOR = {"Time":"2021-03-26T16:34:11","ENERGY":{"TotalStartTime":"20
16:34:41 RSL: sonoff-pow1/tele/STATE = {"Time":"2021-03-26T16:34:41","Uptime":"0T00:01:39","UptimeSe
16:34:41 RSL: sonoff-pow1/tele/SENSOR = {"Time":"2021-03-26T16:34:41","ENERGY":{"TotalStartTime":"20
16:35:11 RSL: sonoff-pow1/tele/STATE = {"Time":"2021-03-26T16:35:11","Uptime":"0T00:02:09","UptimeSe
16:35:11 RSL: sonoff-pow1/tele/SENSOR = {"Time":"2021-03-26T16:35:11","ENERGY":{"TotalStartTime":"20

```

Figure 6.7: Data log on the Tasmota console

6.5 Storing Smart Meter Data to a Database

Once the smart meter is transmitting data to the MQTT broker at fixed intervals, the next step is to save the data to the database. MySQL database is used to store the data that is received from the smart meters, but the Mosquitto MQTT broker does not have

```

Array
(
  [Time] => 2021-03-26T16:59:19
  [ENERGY] => Array
    (
      [TotalStartTime] => 2020-12-20T15:07:58
      [Total] => 0.074
      [Yesterday] => 0
      [Today] => 0.074
      [Power] => 15
      [ApparentPower] => 29
      [ReactivePower] => 26
      [Factor] => 0.5
      [Voltage] => 237
      [Current] => 0.124
    )
  )
)

```

Figure 6.8: Screenshot of data sent by the smart meter

the functionality to add data that is received from publishers to a database. This has to be done on the client-side of the MQTT architecture. The client has to be able to subscribe to a particular topic on the broker and receive messages from a publisher. The client must also be able to save this data to a database when it receives it.

A suitable client for this purpose is the Node-Red application. Node-Red is a web-based development tool for visual programming, and it uses flows to develop programs for linking APIs, online services, and hardware devices [122]. The first thing to do on the client is to add the same MQTT broker as the one on the smart meter. If the client connects to the broker that means it can now publish messages to that broker and subscribe to topics sent to the broker from other publishers. Figure 6.9 below shows the Node-Red flow for listening for messages from the smart meter.

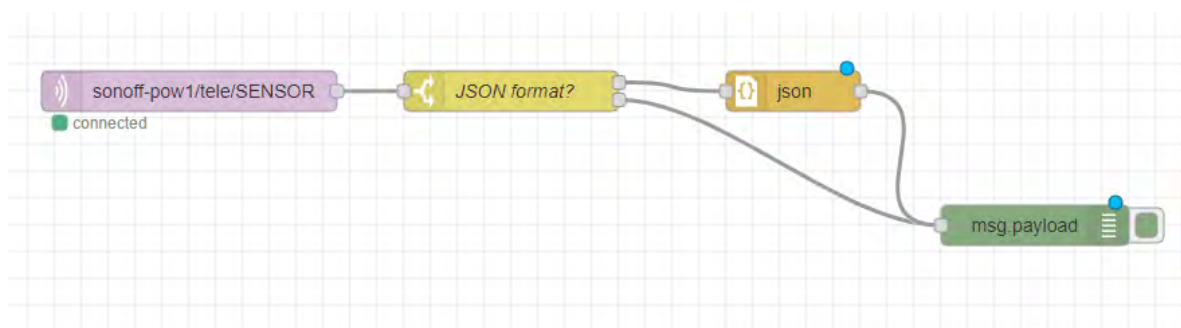


Figure 6.9: Node-Red flow for subscribing to smart meter MQTT topic

The MQTT topic that the Node-Red client has subscribed to is the same topic that was specified on the smart meter, which means the client can now receive messages that

are sent by the smart meter through the `sonoff-pow1` topic. The next part of the flow is to check if the data that has been received is in JSON format. If it is not, then it is first converted to JSON format before it is printed to the Node-Red console. Figure 6.10 below shows the console of the Node-Red client when the smart meter is configured to transmit data every 60 seconds.



The screenshot displays the Node-Red console with five entries, each representing a received MQTT message. Each entry includes a timestamp, the node ID (7c466513.0a270c), the topic (`sonoff-pow1/tele/SENSOR`), and the message payload. The payload is a JSON object containing a `Time` string and an `ENERGY` object.

```
sonoff-pow1/tele/SENSOR : msg.payload : Object
  { Time: "2021-03-29T12:50:05",
    ENERGY: object }

3/29/2021, 1:51:06 PM node: 7c466513.0a270c
sonoff-pow1/tele/SENSOR : msg.payload : Object
  { Time: "2021-03-29T12:51:05",
    ENERGY: object }

3/29/2021, 1:52:06 PM node: 7c466513.0a270c
sonoff-pow1/tele/SENSOR : msg.payload : Object
  { Time: "2021-03-29T12:52:05",
    ENERGY: object }

3/29/2021, 1:53:06 PM node: 7c466513.0a270c
sonoff-pow1/tele/SENSOR : msg.payload : Object
  { Time: "2021-03-29T12:53:05",
    ENERGY: object }

3/29/2021, 1:54:06 PM node: 7c466513.0a270c
sonoff-pow1/tele/SENSOR : msg.payload : Object
  { Time: "2021-03-29T12:54:05",
    ENERGY: object }
```

Figure 6.10: Console of Node-Red client subscribed to smart meter MQTT topic

The data is being successfully received by the Node-Red client, and the next step is to save it to the database. The database, in this instance, serves the simple purpose of storing how much energy a user has used, how much energy they have left, and the time this information was last updated. This is to enable a potential buyer to have enough information about whether or not a seller has enough energy to complete the transaction. The Node-Red flow for saving data to a MySQL database is shown in Figure 6.11.

The flow to add the smart meter data to the database first receives the data in a message from the MQTT broker using the `sonoff-pow1` topic. The data is then converted to JSON format before it is passed through a function that contains the SQL statement to update the database with the given values. This flow is executed every time the smart meter sends data to the MQTT broker. Each smart meter in the system has its own

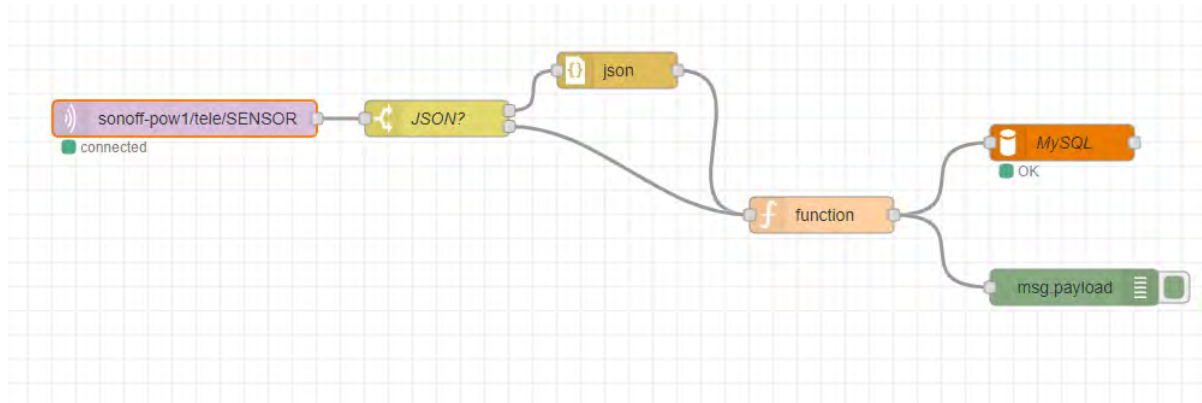


Figure 6.11: Node-Red flow for saving data to a database

flow in the Node-Red client, as shown in Figure 6.12 below. The different flows represent the different topics that the client is subscribed to. Each smart meter has a different topic name which acts as a unique identifier for the smart meter in the system and is used by the Node-Red client to know which record to update in the database. When a particular smart meter sends an update to the MQTT broker, the Node-red client receives it through the flow that is subscribed to that topic and updates the record in the database corresponding to that smart meter.

6.6 Energy Transfer

This section details the physical transfer of energy from the seller's battery to the buyer's battery when the buyer initiates a purchase on the web-based application. Each node consists of a battery, a Raspberry Pi microcontroller, and a relay for every other node that is in the system. Figure 6.13 below shows how the components are connected to each other within a single node as part of a system with two other nodes.

The battery powers the microcontroller. The first smart meter is connected to the battery's output terminals to measure the current that is leaving the battery to the load or to other nodes during a transaction. The second smart meter is connected to the battery's input terminal so that it measures the energy that goes into the battery when energy is purchased from another node. The relays each have a power, ground, and signal terminal. These terminals are connected to the GPIO pins on the Raspberry Pi. The relay's power terminal is connected to one of the two 5V pins on the Raspberry Pi, and

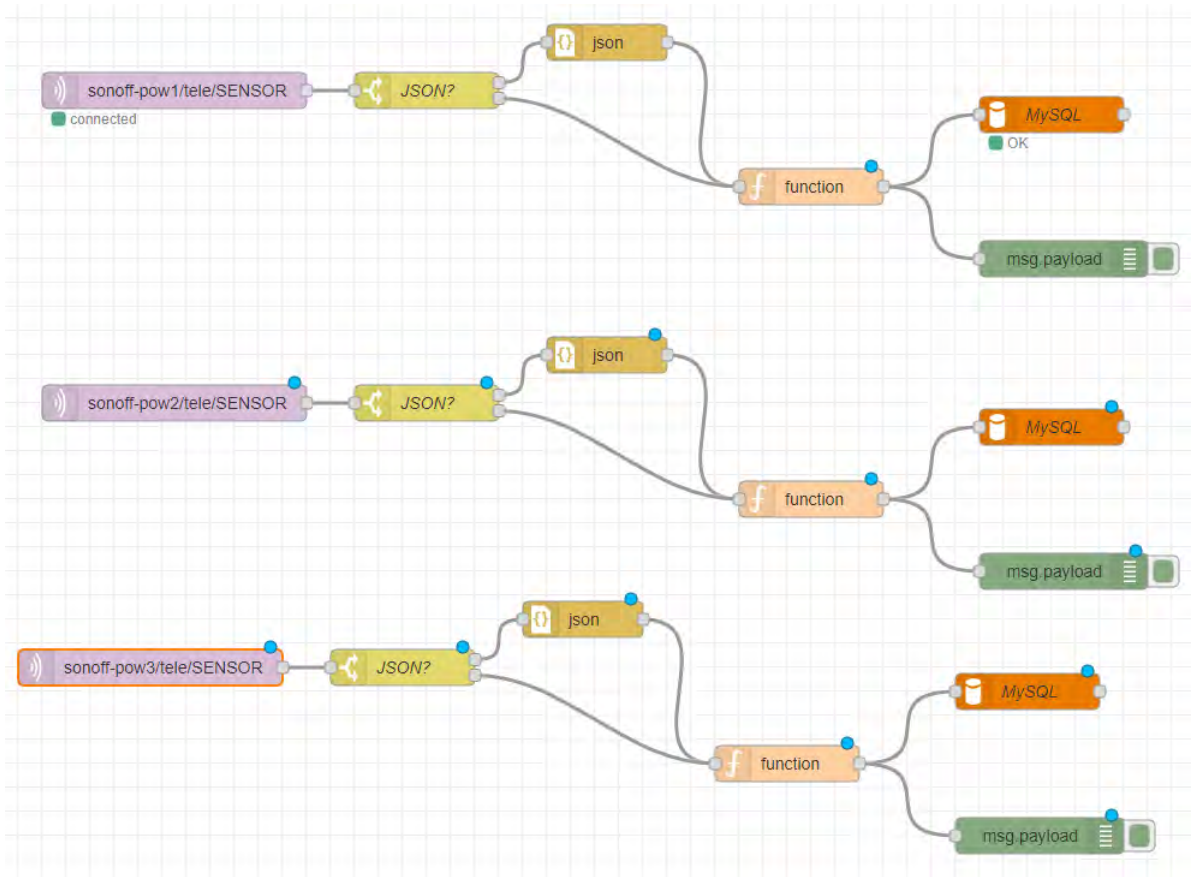


Figure 6.12: Multiple Node-Red flows for saving data from multiple smart meters to a database

the ground terminal of the relay is connected to a ground pin on the Raspberry Pi. The signal terminal is connected to one of the digital pins on the Arduino. Different relays can share the same 5V and ground pins on the Raspberry pi, as illustrated in Figure 6.14. This is because the relays are activated at different times, and so only one relay draws power from the 5V pin at a time.

Each node's microcontroller has a script that opens the appropriate relay and transfers energy to the battery that is connected to that relay. Each node has a specific GPIO pin, and every other node uses that pin to identify that node. An example is a node using GPIO pin 23. This means that the signal terminal on relays connected to that node will be connected to that pin on every other node. In the MySQL database, there is a field that contains a node's GPIO pin so that the web application can identify that node and transfer energy to it. When a buyer purchases energy from a seller, a command is sent to the seller's microcontroller to open the relay that is connected to the buyer's

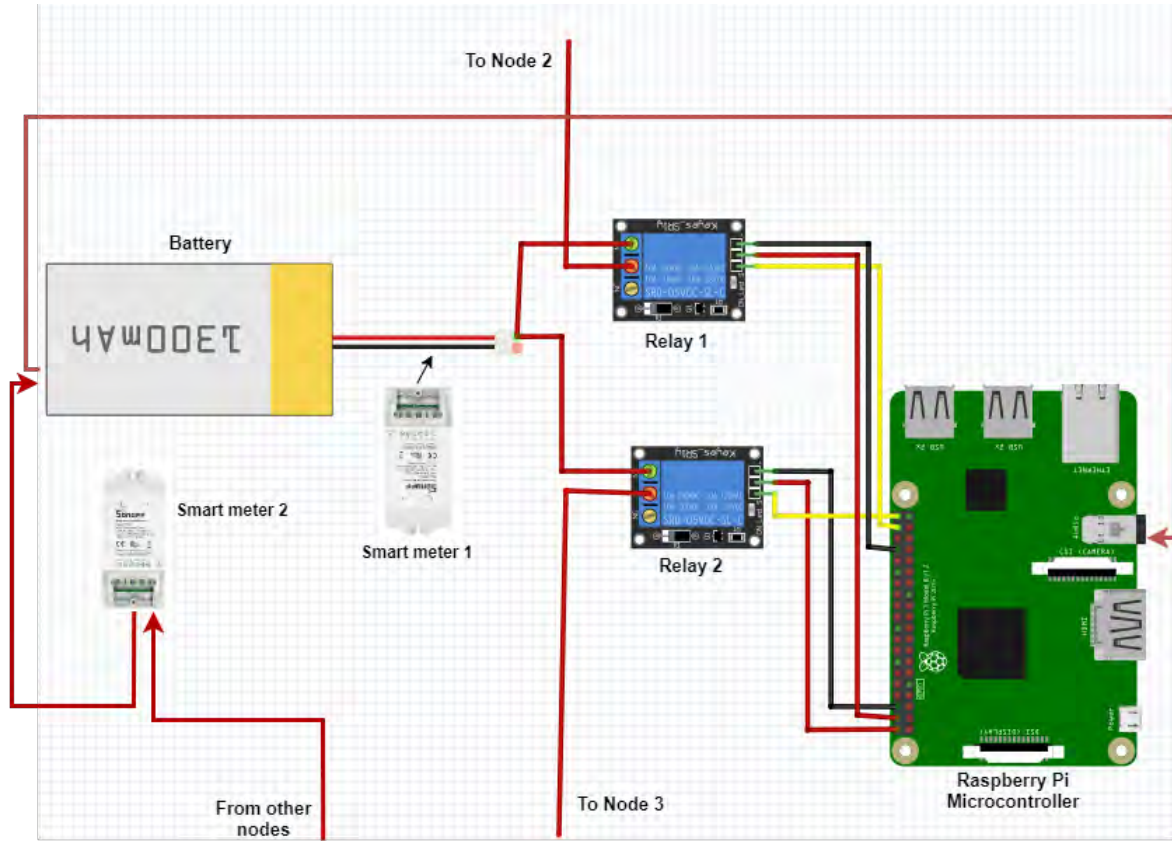


Figure 6.13: Connection of components on a single node

battery and allow energy to flow from the seller's battery to the buyer's battery. This command passes two values to the script. It passes the GPIO pin number that the relay is connected to as well as the amount of energy to transfer.

6.7 Measuring Energy Transferred

To reduce the complexity of the prototype system, a simple method is used to measure the amount of energy that is transferred from the seller's battery to the buyer's battery. The known output amperage of the battery is used together with measuring the connection time to calculate the amount of energy that has been transferred. This calculation is done using the formula for charge which is shown below, where q is the charge in Ah, I is the current in Amps, and T is the time in hours.

$$q = IT \quad (6.4)$$

A battery with an output of 1.2A transfers 1200mAh of energy in an hour, and so if a

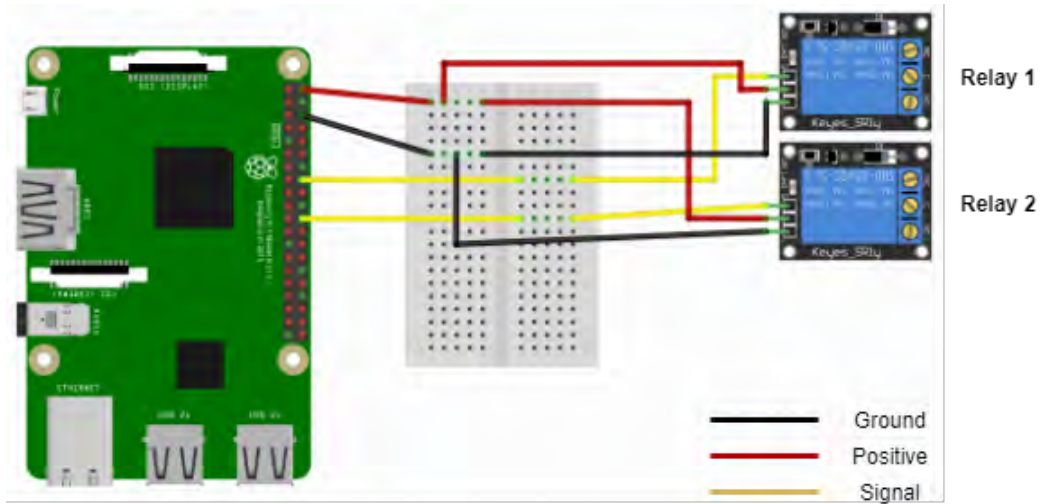


Figure 6.14: Connection of multiple relays to the same ground and 5V terminals

buyer purchases 200mAh of energy, then the relay connecting the seller's battery output and input of the buyer's battery will have to be open for 10 minutes using the calculation below.

$$q = IT$$

$$T = \frac{q}{I}$$

$$T = \frac{0.2}{1.2}$$

$$T = \frac{1}{6} \text{ hours}$$

$$t = \frac{1}{6}(60)$$

$$t = 10 \text{ minutes}$$

The value of t is converted to seconds first and is then passed by the web application to the script on the seller's microcontroller. The script is then executed on the microcontroller, and the transfer of energy begins. The transfer of energy goes on for the amount of time that is specified when the script is executed. Once the time has elapsed, the microcontroller closes the relay, and the transfer of energy stops.

6.8 Checking Transaction Status

On the buyer's side, the smart meter records the energy that is coming in and sends that data to the database every 30 seconds during the transaction. After the specified time has elapsed, the system checks how much energy passed through the buyer's smart meter and compares that figure to the expected amount of energy that was supposed to be transferred. During the transfer of energy, some of it is lost as it is converted to heat in the conductors connecting the two points, and so this is taken into account when the system checks if the transaction was completed successfully. Due to the short distance between the seller and the buyer's battery in the prototype, the acceptable level of energy losses is 5% which means that if the buyer's battery receives less than 95% of the energy that they purchased, then that transaction is considered to be incomplete and the buyer receives a refund for the energy that they did not receive. In the previous example, where a buyer purchases 200mAh from a seller, it means that the buyer has to receive at least 95% of the energy they purchased for the transaction to be considered successful. That equates to 190mAh, and if the buyer receives less than that in the specified 10 minutes, then the system assumes that something went wrong during the transfer, and the buyer will only pay for the energy that they received.

6.9 Conclusion

The hardware components of the peer-to-peer energy trading system have been identified and set up. They have been individually configured and tested and the smart meter has been successfully tested. The next step is to test the system as a whole with multiple nodes to see if it performs as it should.

Chapter 7

System Testing

The main aim of this chapter is to conduct a series of tests on the peer-to-peer energy trading platform under various conditions to see if it functions as intended and that the various components of the system have been integrated in a correct manner.

7.1 Test Set-up

This section details the blockchain and hardware set-ups for the tests to be conducted. The test set-up consists of three nodes that have different roles in the system. The web-based application and database are deployed to a separate computer that simulates the function of a cloud-based service provider. This computer does not have any other functions in the system except to host the database and the application.

7.1.1 Blockchain Network Set-up

Model	Raspberry Pi 3 Model B+
Processor	Broadcom Cortex-A53 64-bit @ 1.4GHz
Memory	1GB LPDDR2 SRAM
Connectivity	2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN
SD Card	64 GB Micro SD class 10 @ 95MB/s read speed
Input Power	5V/2.5A DC

Table 7.1: Raspberry Pi 3 Model B+ Specifications [123]

The blockchain network that was created previously was used, and it had three nodes that were part of the network. The network uses the proof of authority consensus mechanism with a block creation period of 15 seconds. Two out of the three nodes were sealer

nodes meaning they were able to validate blocks of transactions. All the three nodes were running on Raspberry Pi computers, whose technical specifications are shown in Table 7.1 above.

Each node had one account on it, and the public addresses of these nodes are shown below, together with the role of the node in the network and its listening port.

Node 1

- Ethereum Address: 0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29
- Port: 3011
- Role: Sealer node

Node 2

- Ethereum Address: 0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69
- Port: 3012
- Role: Sealer node

Node 3

- Ethereum Address: 0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01
- Port: 30303
- Role: None

7.1.2 Hardware Set-up

The hardware configuration consists of three nodes, namely Node 1, Node 2, and Node 3, as shown in Figure 7.1. Each node has a Raspberry Pi microcontroller, a battery, two relays, and two smart meters. The relays are connected to each of the other nodes in the system. One smart meter is connected between a node's battery, and its microcontroller, which acts as the load, and the other smart meter is connected between the node's battery and relays from other nodes. The second smart meter is used to verify that a node receives the correct amount of energy during a transaction. Each node has a dedicated GPIO pin

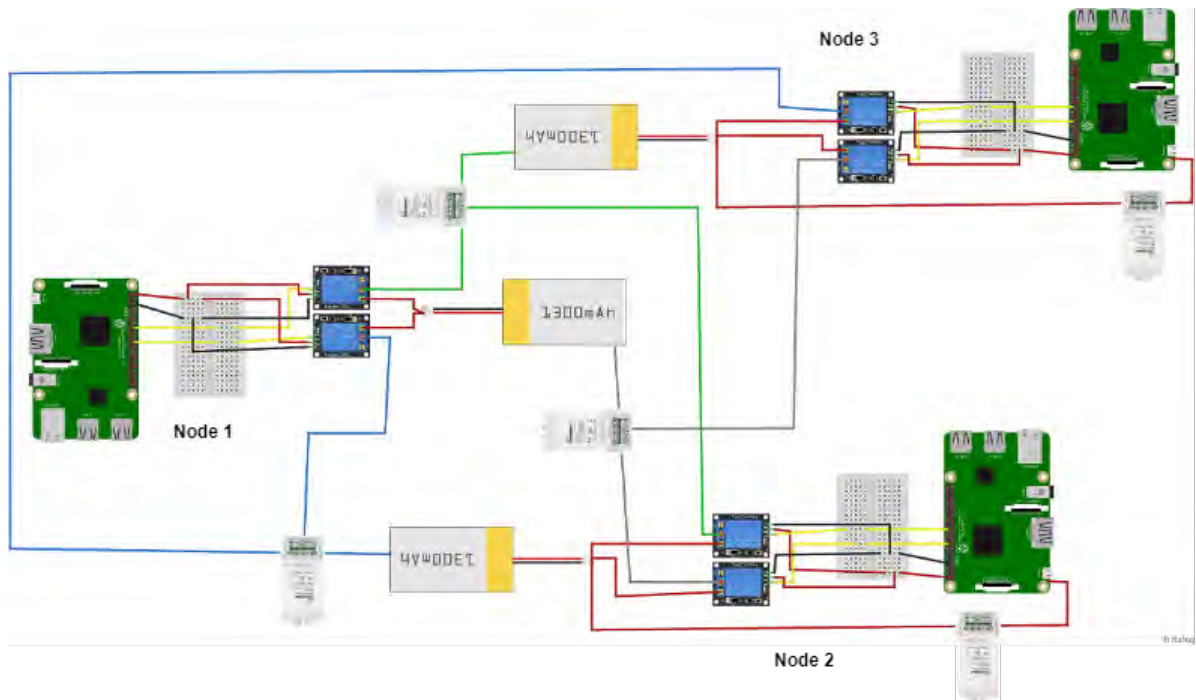


Figure 7.1: Hardware set-up of three nodes

in the system which means that the signal terminal of all the relays that lead to a node's battery are connected to the same GPIO pin. An example is Node 1 that uses pin 12, and so the relays that lead to Node 1's battery on both Node 2 and Node 3 are connected to GPIO pin 12 on their microcontrollers. Each node's GPIO pin and the capacity and output current of their batteries is shown below.

Node 1

- GPIO pin: 12
- Battery Capacity: 5000mAh
- Current Output: 2.1A

Node 2

- GPIO pin: 16
- Battery Capacity: 1200mAh
- Current Output: 1A

Node 3

- GPIO pin: 18
- Battery Capacity: 1200mAh
- Current Output: 1A

7.2 Adding New Listing

The first test is for adding a new listing to the system. There is already one listing that was added by Node 1, as shown in Figure 7.2. Node 2 intends to sell 150mAh of energy for 3 Eth, and so they use the web application to list the energy.

Peer to Peer Energy Trading

0xED5d358F28E4Ae971ECa8e2Da7DC5677d177aa69

Add New Listing

Add Listing

Buy Electricity

Listing ID	Quantity	Price	Seller	
4	200 mAh	2 Eth	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	<div>Buy</div> <div>Remove</div>

Sellers' Electricity Balances

Seller	Current Balance	Last Updated
0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	4887 mAh	2021-03-30T17:16:04

Figure 7.2: Adding new listing through the web-based application

Since there is already a listing on the application, the current electricity balance of the seller is retrieved from the database and displayed on the application. Once the user on Node 2 clicks on the “Add Listing” button, a Metamask window pops up, as shown in Figure 7.3. The Metamask window shows the type of transaction that the user is trying to initiate, which in this case is a contract interaction meaning it is a smart contract call. The window also shows the transaction cost that the user must pay to add the listing to the smart contract.

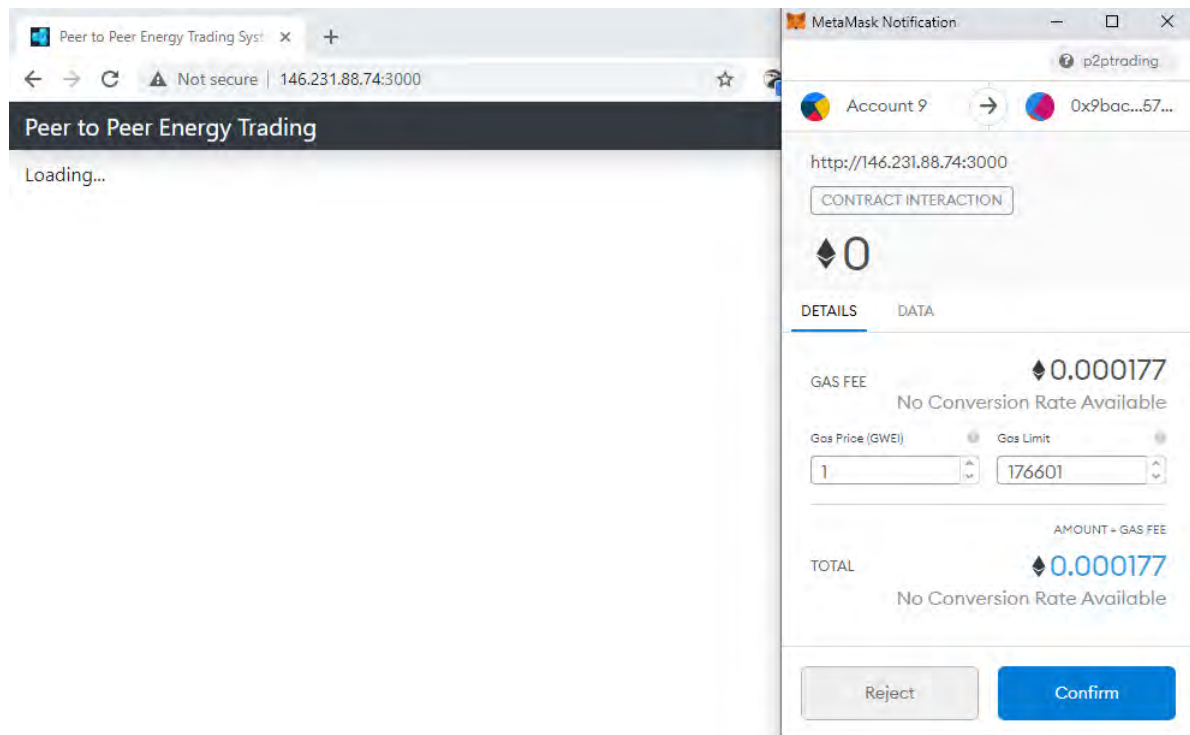


Figure 7.3: Transaction fee for adding a new listing

Once the user clicks on confirm, the transaction is initiated on the blockchain. That means that the transaction is added to a candidate block and waits to be validated. The waiting time in this instance can be up to 15 seconds since that is the block creation interval that was specified in the genesis file. When the block containing the transaction has been validated, Metamask sends a notification to the user through the browser, as shown in Figure 7.4.

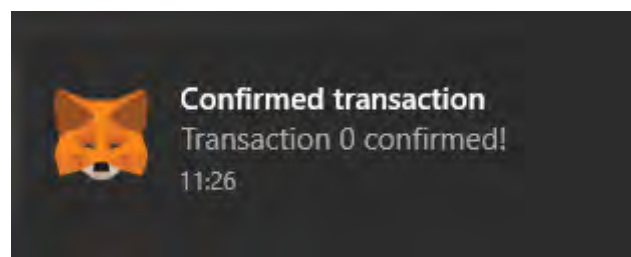


Figure 7.4: Browser notification from Metamask after the transaction is added to the blockchain

When the transaction was confirmed, the new listing appeared on the web-based application as shown in Figure 7.5. Like the first listing, the new listing also has details such as the address of the seller, a listing ID, the quantity and price of the energy,

and options to buy and delete the listing. Since it was Node 2's first listing, the system retrieved its electricity balance from the database and displayed it on the web application so that potential buyers could see whether or not the seller had enough energy in its battery to complete the transaction.

Peer to Peer Energy Trading0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69

Add New Listing

Electricity Quantity (mAh)

Electricity Price (Eth)

Add Listing

Buy Electricity

Listing ID	Quantity	Price	Seller		
4	200 mAh	2 Eth	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	Buy	Remove
5	150 mAh	3 Eth	0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69	Buy	Remove

Sellers' Electricity Balances

Seller	Current Balance	Last Updated
0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	4879 mAh	2021-03-30T17:31:34
0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69	894 mAh	2021-03-30T17:32:27

Figure 7.5: Web-based application after the listing is added

7.3 Making a Purchase

This section involves testing the client-side application by making a series of purchases of energy from the web-based application. The first transaction was not interrupted so as to see how the system handles such a transaction, and the second one was interrupted to see how the system works under those conditions.

7.3.1 Complete Transaction

The first transaction involves Node 1 purchasing the energy listed by Node 2. To verify the amount of energy that is transferred during the transaction, the starting values need

to be known. Figure 7.6 below shows a screenshot of the database table that shows the amount of energy that is each node's battery before the start of the transaction.

ClientID	Address	Time	Today	Total	Remaining
sonoff-pow1	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	2021-03-31T10:05:37	0.419	0.419	4845
sonoff-pow2	0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69	2021-03-31T10:05:06	0.43	0.43	852
sonoff-pow3	0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01	2021-03-31T10:04:56	0.21	0.21	1021

Figure 7.6: Database table showing the energy balances of all three nodes

Node 1, which is identified by the name of its MQTT topic sonoff-pow1, has 4845mAh of energy in its battery before the transaction, and the seller, which is Node 2, has 852mAh of energy. Node 1 initiates the transaction by clicking on the “Buy” button for the energy that was listed by Node 2. This action prompts the system to start the transfer of energy from the seller's battery to the buyer's battery. The application retrieves the buyer's GPIO pin number from the database as well as the seller's IP address and the current output of their battery. The application then calculates the amount of time that the transfer of energy will take using the formula $q = IT$, and in this case, the time is 9 minutes. The application then sends a command to the seller's microcontroller to open the relay on the specified GPIO pin, which in this case is pin 12. The amount of time that the relay should be open for is also passed as one of the parameters.

Buy Electricity

Listing ID	Quantity	Price	Seller		
4	200 mAh	2 Eth	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	Buy	Remove
5	150 mAh	3 Eth	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29		Remove

Sellers' Electricity Balances

Seller	Current Balance	Last Updated
0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	4992 mAh	2021-03-31T10:16:14

Figure 7.7: Web application after successful transaction

After the specified time has passed, the system then retrieves the updated energy balances for both the buyer and the seller and compares them with how much energy they

should have after a successful transaction. Both figures are in the acceptable range, as shown in Figure 7.7 above, and so the transaction is considered successful by the system, which then transfers the 3 Ether plus transaction charges from the buyer’s account into the seller’s account. A screenshot of Metamask showing the details of the transaction is shown in Figure 7.8 below. The smart contract also changes the value of the owner of the listing from the seller to the buyer, and so the address of the buyer is the one that now appears on the listing’s details in the application. This is to enable the buyer to be able to delete the listing as only the owner of the listing can delete it. The “Buy” button also disappears as the energy has been purchased.

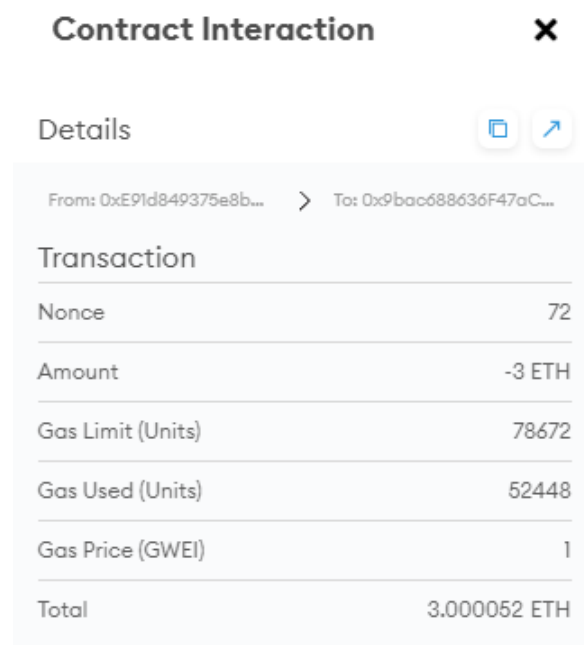


Figure 7.8: Metamask screenshot showing transaction details

7.3.2 Incomplete Transaction

The test for an incomplete transaction involves Node 3 purchasing the energy listed by Node 1, but unlike the previous transaction, the cable connecting Node 1’s battery to Node 3’s battery will be disconnected during the transfer of energy. This is to see how the system handles such a situation. Node 3 will purchase the 200mAh of energy listed by Node 1, and the amount of energy in both batteries before the transaction is shown in Figure 7.9.

ClientID	Address	Time	Today	Total	Remaining
sonoff-pow1	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	2021-04-01T18:23:46	1.35	1.35	4695
sonoff-pow2	0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69	2021-04-01T18:22:49	0.749	0.749	1048
sonoff-pow3	0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01	2021-04-01T18:23:17	2.145	2.145	335

Figure 7.9: Database table showing energy balances of all the nodes before the start of a transaction

Node 1's battery has a current output of 2.1A which means it will take 5 minutes and 42 seconds to transfer 200mAh of energy to Node 3's battery. For the transaction to be considered successful, that means that at least 95% of the 200mAh has to be received by the buyer from the seller. If this does not happen, then the transaction is considered to have failed, and the seller does not receive the full amount for the energy. Instead, the system should calculate how much is due to the seller based on how much energy was successfully transferred.

In this test, Node 3 will make the purchase as before, but the transfer of energy will be interrupted exactly 2 minutes after it begins. The time that the energy is transferred is known in this instance so as to have a known outcome that the system should produce. If the relay is open for exactly 2 minutes, then the amount of energy that is transferred in that period from a battery with an output current of 2.1A is calculated below.

$$q = IT$$

$$q = (2.1) \frac{2}{60}$$

$$q = \frac{2.1}{30}$$

$$q = 0.07$$

$$q = 70 \text{ mAh}$$

This means that 70mAh is transferred in that period out of the total figure of 200mAh. Instead of transferring the full 2 Eth to the seller, the system will calculate the amount due to the seller from the amount of energy that was transferred. This calculation is shown below.

$$\begin{aligned}
\text{Amount due} &= \frac{\text{energy transferred}}{\text{total energy due}} (\text{original price}) \\
&= \frac{70}{200} (2) \\
&= 0.70 \text{ Eth}
\end{aligned}$$

This means that the smart contract has to transfer 0.70 Eth to the seller after the transaction interval has passed. Now that the desired outcomes are known, the next step is to carry out the actual test and observe the results.

ClientID	Address	Time	Today	Total	Remaining
sonoff-pow1	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	2021-04-01T18:29:59	1.374	1.374	4621
sonoff-pow2	0xED5d358F2BE4Ae971ECa8e2Da7DC5677d177aa69	2021-04-01T18:29:39	0.753	0.753	1013
sonoff-pow3	0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01	2021-04-01T18:30:06	2.152	2.152	398

Figure 7.10: Database table showing energy balances 6 minutes after transaction started

Figure 7.10 above shows a screenshot of the database table that contains the energy balances approximately 6 minutes after the transaction was initiated. From the balances of the two nodes involved in the transaction, it is clear to see that not the full 200mAh was transferred from Node 1 to Node 3. Therefore not the full amount was due to the seller since the transaction was not completed.

Transaction	
Nonce	73
Amount	-0.7 ETH
Gas Limit (Units)	78672
Gas Used (Units)	52448
Gas Price (GWEI)	1
Total	0.700052 ETH

Figure 7.11: Metamask screenshot showing the amount paid to the seller and transaction cost

The system calculates the amount due to Node 1, and in this case, it is 0.7 Eth. This amount is passed to the smart contract call, which then initiates the transfer of funds

from Node 3's account to Node 1. The Metamask transaction receipt is shown in Figure 7.11. The smart contract transfers the amount owed to the seller and also deducts the transaction cost from the buyer's account.

Figure 7.12 shows the web application after the incomplete transaction has been concluded. The smart contract function that handles incomplete transactions receives the amount that is due to the seller and updates the price on the listing. The status of the listing is also changed to show that it has been purchased, and so the buy button disappears from the listing. The address of the owner of the listing changes to reflect the address of the buyer. The electricity balance of Node 1 also disappears from the web application since they do not have any listings on the application.

Buy Electricity

Listing ID	Quantity	Price	Seller	
4	200 mAh	0.7 Eth	0x86CE3b9540eD1727D55336baeaB70Aa8d5d02A01	<button>Remove</button>
5	150 mAh	3 Eth	0xE91d849375e8bA9c0b83b2c8006dd4EC3cA71A29	<button>Remove</button>

Sellers' Electricity Balances

Seller	Current Balance	Last Updated
--------	-----------------	--------------

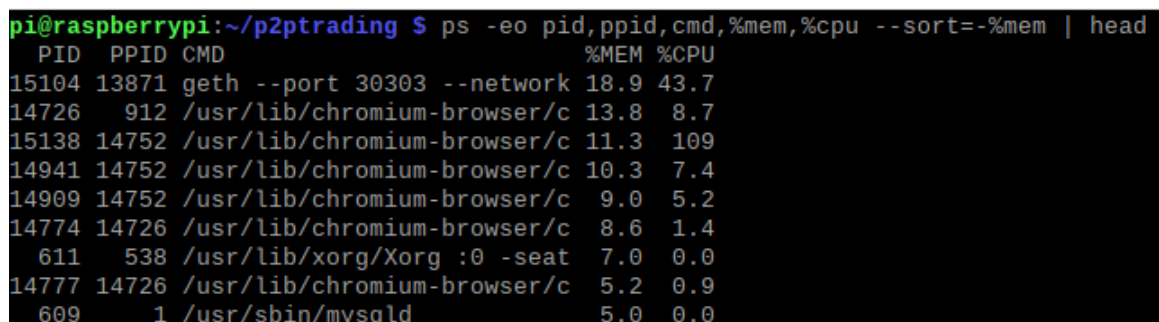
Figure 7.12: Web-based application after conclusion of the transaction

7.4 PoA-based Blockchain Resource Usage

This section looks at the amount of computer resources that the blockchain network uses to demonstrate that the system can be run on low-specification computers like a Raspberry Pi. To show this, the amount of RAM and the portion of the processor being used by the peer-to-peer energy trading system were observed.

7.4.1 Resource Usage by non-Sealer Node

The first test was conducted on a non-sealer node to see how much computer resources the node uses when the system is running. In the test set-up there is only one non-sealer node which is Node 3 and so that node was used to conduct this test.



PID	PPID	CMD	%MEM	%CPU
15104	13871	geth --port 30303 --network	18.9	43.7
14726	912	/usr/lib/chromium-browser/c	13.8	8.7
15138	14752	/usr/lib/chromium-browser/c	11.3	109
14941	14752	/usr/lib/chromium-browser/c	10.3	7.4
14909	14752	/usr/lib/chromium-browser/c	9.0	5.2
14774	14726	/usr/lib/chromium-browser/c	8.6	1.4
611	538	/usr/lib/xorg/Xorg :0 -seat	7.0	0.0
14777	14726	/usr/lib/chromium-browser/c	5.2	0.9
609	1	/usr/sbin/mysqld	5.0	0.0

Figure 7.13: Resource usage on Node 3

Figure 7.13 shows the processes that were running on Node 3 during the test. The blockchain network used the most resources out of all the processes but the actual amount of computer resources that it was using were still very low. It used almost 19% of RAM which amounts to approximately 190 MB of RAM. This is a low figure by modern standards and the proportion of CPU usage seems high but that is because the test computer only had a 1.4GHz processor which is lower than what modern smart phones have. A Chromium browser was also running on the computer and it had a single tab open that was running the web-based application.

7.4.2 Resource Usage by Sealer Node

The next step was to check how much computer resources are used by a sealer node. The most important thing to check for here was how much extra resources it takes to validate blocks.

Figure 7.14 shows the resource usage on Node 1 which was used to conduct the test. The major observation was that the blockchain network uses more memory on sealer nodes than on non-sealer nodes. The memory usage on sealer nodes was approximately double that of non-sealer nodes with a range of between 350 MB and 450 MB. This is very low for a full node that validates blocks because the minimum requirements for the PoW-

```

pi@raspberrypi:~ $ ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
  PID  PPID  CMD                                %MEM %CPU
14460 13871 geth --port 3011 --networki 38.5 38.5
14093   912 /usr/lib/chromium-browser/c 12.8 11.4
14316 14117 /usr/lib/chromium-browser/c   9.8 12.5
14139 14093 /usr/lib/chromium-browser/c   8.4  1.1
14298 14117 /usr/lib/chromium-browser/c   8.4  6.9
   611   538 /usr/lib/xorg/Xorg :0 -seat  7.3  0.0
14237 14117 /usr/lib/chromium-browser/c   6.8  1.0
   609    1 /usr/sbin/mysqld              5.2  0.0
14143 14093 /usr/lib/chromium-browser/c   4.5  1.1

```

Figure 7.14: Resource usage on a sealer node

based Ethereum Mainnet are at least 4 GB of RAM [124]. This means that a PoA-based blockchain network uses significantly less resources than a network that uses the PoW consensus mechanism. The CPU usage for a sealer node did not change from that of a non-sealer node. The variance between Node 1 and Node 3 during the test was within an acceptable range to conclude that there was no difference. This is because even though Node 1 validates blocks, this validation does not involve any complex computations and so there is very little extra CPU usage during that process.

7.4.3 Resource Usage during Energy Transfer

The last test was to see how much extra computer resources are required during the transfer of energy from one battery to another. The only additional process that was running during this time was a script on the seller's computer that opened the relay to allow energy to flow to the buyer's battery.

```

pi@raspberrypi:~ $ ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
  PID  PPID  CMD                                %MEM %CPU
16731 13871 geth --port 3011 --networki 40.7 36.6
16374   912 /usr/lib/chromium-browser/c 11.9  9.7
16535 16397 /usr/lib/chromium-browser/c   9.4  4.3
16561 16397 /usr/lib/chromium-browser/c   9.1  9.6
16418 16374 /usr/lib/chromium-browser/c   8.5  2.1
16701 16397 /usr/lib/chromium-browser/c   6.9  0.8
   611   538 /usr/lib/xorg/Xorg :0 -seat  5.4  0.0
16422 16374 /usr/lib/chromium-browser/c   4.9  0.8
16395 16374 /usr/lib/chromium-browser/c   3.8  0.1

```

Figure 7.15: Resource usage on node 1 during transfer of energy

To initiate the transfer of energy, the Raspberry Pi sends a digital signal through the appropriate GPIO pin to the relay to turn it on and then sends another signal to turn

it off. This process uses very little computer resources and does not even appear on the console when the list of currently running processes is retrieved as illustrated in Figure 7.15. This test showed that all the components of the peer-to-peer energy trading system can be run simultaneously on a low-end computer such as a Raspberry Pi.

7.5 Conclusion

The blockchain-based peer-to-peer energy trading system has been successfully integrated and tested. Various tests using different scenarios that were supposed to produce different outcomes were tested and the results were as expected. The amount of computer resources used by the system were also observed.

Chapter 8

Conclusion

This research aimed to come up with a way to decentralize the market for excess energy generated from renewable energy sources by consumers. This was done through the use of blockchain technology and the internet of things to come up with a system that significantly reduces the utility company's role and allows the participants to transact with each other directly. The research gave a detailed overview of what blockchain is, how transactions are handled in a blockchain and how to create a private blockchain network using Ethereum. In this chapter, the initial objectives of the research are revisited and an evaluation is done on if and how those objectives were met. The chapter also looks at how the research can be expanded under future work and concludes the whole thesis by summarizing the work carried out.

8.1 Achieved Objectives

This section evaluates how the initial objectives were met during the course of the research. The objectives as set out in section 1.4 of the study are as follows:

1. Investigate the possibility of the use of a low-resource blockchain configuration to create a blockchain network that maintains the other properties of blockchain technology.
2. Use IoT to gather information on how much energy has been used by an entity over a period of time.
3. Create a prototype for a blockchain-based peer-to-peer energy trading system.

8.1.1 Low-resource Blockchain Configuration

The amount of computer resources and energy that is used by a blockchain network is determined by the consensus mechanism used by the network. A number of consensus mechanisms were evaluated, and they were compared under various criteria, such as their level of decentralization, privacy, and the resources they require. The PoA consensus mechanism was chosen as the preferred consensus mechanism for the blockchain network for the proposed system as it does not require a lot of processing power to validate transactions. The blockchain network was deployed on Raspberry Pi computers with only a 1.4GHz processor and 1GB of RAM, which are very low specifications for a full node on a blockchain. The blockchain network was tested and shown to not use a lot of computer resources.

8.1.2 Data Collection using IoT

The next objective was to collect data from nodes about how much energy they use over time and how much energy they have left in their batteries. A smart meter was used for this purpose, and it was first configured with a firmware that best makes use of its features and collects information relevant to the proposed system. A question that arose during the process of trying to meet this objective was the appropriate data transfer protocol to use to send data from the smart meter to the cloud-based database. The ideal data transfer protocol needed to have a low bandwidth footprint as the data would need to be transferred at regular intervals. MQTT was chosen for this purpose due to its speed and the small packets of data that it transmits.

8.1.3 Peer-to-peer Energy Trading System Prototype

The primary objective built on the previous objectives by coming up with a prototype for a system that allows participants to buy and sell excess energy amongst each other. The system consisted of an Ethereum-based private blockchain network, in which all participants in the system were represented by nodes on the network. The blockchain network was used to facilitate payments using the native Ether cryptocurrency, and a

smart contract was also deployed to it. The smart contract holds the logic for when and how a node should be paid. Smart meters sent the collected data to a cloud-based storage facility from where a web-based application retrieved the data. The web-based application provided an interface for the transactions to be made by the buyers and sellers, and it used the data from the smart meter to let buyers know if they had the energy that they intended to sell.

The last part of the prototype was the physical components that were connected to each other to demonstrate the transfer of energy from a seller to a buyer. Relays were used to start and stop the transfer of energy. The Raspberry Pi computers that were used as nodes in the blockchain network were also used as microcontrollers for the relays. This reduced the number of physical components in the system. The prototype was successfully tested, and the system managed to handle the different test scenarios.

8.2 Future Work

The research leaves room for more work to be carried out to either expand the scope or add more features to the prototype. Some work that can be done to expand on this research is explained below.

- The scalability of the prototype can be investigated to see how it would perform in a real-world application with higher voltages and currents.
- The data collected by the smart meter can be used for more applications, such as creating a usage pattern for each user to predict how much energy they require per day. This can then be used to advise users on the amount of energy they should buy or sell.

8.3 Summary

This research has given a detailed background into blockchain technology and used a low-energy, low-computation implementation of blockchain to create a private network. A smart contract was designed and deployed to the blockchain, and all this was integrated

with IoT to create a peer-to-peer energy trading system. The system was tested, and it has been shown that the objectives that this research set out to achieve have been met.

Bibliography

- [1] Arun Ramamurthy and Pramod Jain. “The Internet of Things in the Power Sector: Opportunities in Asia and the Pacific”. In: *ADB Sustainable Development Working Paper Series* 48 (2017), pp. 1–36.
- [2] Anton Eberhard et al. *Underpowered: The State of the Power Sector in Sub-Saharan Africa*. Tech. rep. 2008.
- [3] Jeremy E. J. Streatfeild. “Low Electricity Supply in Sub-Saharan Africa: Causes, Implications, and Remedies”. In: *Journal of International Commerce and Economics* June (2018), pp. 1–16.
- [4] The Economist. *More than half of sub-Saharan Africans lack access to electricity*. Nov. 2019. URL: <https://www.economist.com/graphic-detail/2019/11/13/more-than-half-of-sub-saharan-africans-lack-access-to-electricity>.
- [5] Ali Dorri et al. *LSB: A Lightweight Scalable BlockChain for IoT Security and Privacy*. Tech. rep. New York, 2017.
- [6] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. *Proof-of-Stake Sidechains*. Tech. rep. 2018.
- [7] Dylan Yaga et al. *Blockchain Technology Overview*. Tech. rep. Gaithersburg, 2018.
- [8] Kumar Bhosale et al. “Blockchain based Secure Data Storage”. In: *International Research Journal of Engineering and Technology* 6 (3 2019), pp. 5058–5061.
- [9] Swaathi Kakarla. *An Introduction to the Genesis Block in Ethereum*. 2018. URL: <https://www.skcript.com/svr/genesis-block-ethereum/>.
- [10] L. M. Bach, B. Mihaljevic, and M. Zagar. “Comparative analysis of blockchain consensus algorithms”. In: Opatija: IEEE, 2018.
- [11] Konstantinos Christidis and Michael Devetsikiotis. “Blockchain and Smart Contracts for the Internet of Things”. In: *IEEE Access* 4 (1 2016), pp. 2292–2303.
- [12] Sarwar Sayeed and Hector Marco-Gisbert. “Assessing Blockchain Consensus and Security Mechanisms against the 51% Attack”. In: *Applied Sciences* 9 (1788 2019), pp. 1–17.
- [13] Junqin Huang et al. “Towards Secure Industrial IoT: Blockchain System With Credit-Based Consensus Mechanism”. In: *IEEE Transactions on Industrial Informatics* 15 (6 2019), pp. 3680–3689.
- [14] Yuhao Wang et al. “Study of Blockchain’s Consensus Mechanism Based on Credit”. In: *IEEE Access* 7 (2019), pp. 10224–10231.
- [15] Satoshi Nakamoto. *Bitcoin: A Peer-toPeer Electronic Cash System*. Tech. rep. 2008.

-
- [16] Johannes Sedlmeir et al. “The Energy Consumption of Blockchain Technology: Beyond Myth”. In: *Business and Information Systems Engineering* 62 (2020), pp. 599–608.
- [17] Sunny King and Scott Nadal. *PPCoin: Peer-toPeer Crypto-Currency with Proof-of-Stake*. Tech. rep. 2012.
- [18] Stefano De Angelis et al. *PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain*. Tech. rep. Rome, 2018.
- [19] Hasib Anwar. *Public Vs Private Blockchain: How Do They Differ?* Mar. 2020. URL: <https://101blockchains.com/public-vs-private-blockchain/>.
- [20] Toqeer Ali Syed et al. “A Comparative Analysis of Blockchain Architecture and Its Applications: Problems and Recommendations”. In: *IEEE Access* 7 (2019), pp. 176838–176869.
- [21] Dominique Guegan. *Public Blockchain versus Private blockchain*. Tech. rep. Paris, 2017.
- [22] Demiro Massessi. *Public Vs Private Blockchain In A Nutshell*. Dec. 2018. URL: <https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f#:~:text=Public%20blockchains%20are%20decentralised%2C%20no,Blockchain%20is%20a%20permissioned%20blockchain..>
- [23] Prashun Javeri. *Blockchain Architecture*. 2019. URL: <https://medium.com/@prashunjaveri/blockchain-architecture-3f9f1c6dac5e>.
- [24] Shuai Wang et al. “An Overview of Smart Contract: Architecture, Applications, and Future Trends”. In: Changshu: IEEE, 2018.
- [25] Chun Hui Suen. *Blockchain as a network stack*. June 2019. URL: <https://medium.com/kommercetf/blockchain-osi-stack-2f1482595953>.
- [26] Leonardo Maria De Rossi, Gianluca Salviotti, and Nico Abbatemarco. “Towards a Comprehensive Blockchain Architecture Continuum”. In: Hawaii: IEEE, 2019.
- [27] Claus Dieksmeier and Peter Seele. “Blockchain and business ethics”. In: Lugano: Wiley, 2019.
- [28] Neeraj Agrawal. *Why ransomware criminals use Bitcoin and why that could be their undoing*. 2017. URL: <https://coincenter.org/link/why-ransomware-criminals-use-bitcoin-and-why-that-could-be-their-undoing>.
- [29] Nicolas Christin. “Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace”. In: Rio de Janeiro: ACM, 2013.
- [30] Jack Morse. *Augur protocol leads to Ethereum-based assassination market*. 2018. URL: <https://mashable.com/article/augur-ethereum-blockchain-assassination-market/>.
- [31] Ray King. *What is a Smart Contract and How Does it Work?* 2019. URL: <https://www.bitdegree.org/tutorials/what-is-a-smart-contract/>.
- [32] Stefano Bistarelli et al. “Ethereum smart contracts: Analysis and statistics of their source code and opcodes”. In: *Internet of Things* 11 (2020).
- [33] Petar Tsankov et al. “Securify: Practical Security Analysis of Smart Contracts”. In: Toronto: ACM, 2018.

-
- [34] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. “Digital Supply Chain Transformation toward Blockchain Integration”. In: Hawaii: IEEE, 2017.
 - [35] Zibin Zheng et al. “An overview on smart contracts: Challenges, advances and platforms”. In: *Future Generation Computer Systems* 105 (2020), pp. 475–491.
 - [36] Merit Kolvart, Margus Poola, and Addi Rull. *Smart Contracts*. Ed. by T. Kerikmae and A. Rull. Switzerland: Springer, 2016, pp. 133–147.
 - [37] Shuai Wang et al. “Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends”. In: *IEEE* 49 (11 2019), pp. 2266–2277.
 - [38] Vandana Sharma and Ravi Tiwari. “A review paper on IOT and It’s Smart Applications”. In: *International Journal of Science, Engineering and Technology Research* 5 (2 2016), pp. 472–476.
 - [39] C. Wang and Daneshmand. “Special Issue on Internet of Things (IoT): Architecture, Protocols and Services”. In: *IEEE Sensors Journal* 13 (2013), pp. 3505–3510.
 - [40] Jacob Morgan. *A Simple Explanation of "The Internet Of Things"*. 2014. URL: <https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#3b6b13841d09>.
 - [41] R. Mahmoud et al. “Internet of Things (IoT) security: Current status, challenges and prospective measures”. In: London: ICITST, 2015.
 - [42] Mayuri A. Bhabad and Sudhir Bagade. “Internet of Things: Architecture, Security Issues and Countermeasures”. In: *International Journal of Computer Applications* 125 (14 2015), pp. 1–4.
 - [43] Jie Lin et al. “A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications”. In: *IEEE Internet of Things* 4 (5 2017), pp. 1125–1142.
 - [44] Paul Stokes. *4 Stages of IoT architecture explained in simple words*. 2018. URL: <https://medium.com/datadriveninvestor/4-stages-of-iot-architecture-explained-in-simple-words-b2ea8b4f777f>.
 - [45] Jozef Mocnej et al. “Decentralized IoT Architecture for Efficient Resources Utilisation”. In: *IFAC PapersOnLine* 51 (6 2018), pp. 168–173.
 - [46] Ben Dickson. *Decentralizing IoT networks through blockchain*. 2016. URL: <https://techcrunch.com/2016/06/28/decentralizing-iot-networks-through-blockchain/>.
 - [47] Daniel Minoli and Benedict Occhiogrosso. “Blockchain mechanisms for IoT security”. In: *Internet of Things* 1 (2 2018), pp. 1–13.
 - [48] Avelino Zorzo et al. “Dependable IoT Using Blockchain-Based Technology”. In: Foz do Iguaçu: LADC, 2018.
 - [49] Roshan Raj. *What is Bitcoin Blockchain*. 2019. URL: <https://intellipaat.com/blog/tutorial/blockchain-tutorial/what-is-bitcoins-blockchains/>.
 - [50] Ana Reyna et al. “On Blockchain and its integration with IoT. Challenges and opportunities”. In: *Future Generation Computer Systems* 88 (1 2018), pp. 173–190.

-
- [51] Minhaj A. Khan and Khaled Salah. “IoT Security: Review, blockchain solutions, and open challenges”. In: *Future Generation Computer Systems* 82 (1 2018), pp. 395–411.
- [52] Yao Sun et al. “Blockchain-Enable Wireless Internet of Things: Performance Analysis and Optimal Communication Node Deployment”. In: *IEEE Internet of Things Journal* 6 (3 2019), pp. 5791–5802.
- [53] Lei Hang and Do-Hyeun Kim. “Design and Implementation of an Integrated IoT Blockchain Platform for Sensing Data Integrity”. In: *Sensors* 19 (2228 2019), pp. 1–26.
- [54] Matt Zand. *An Introduction to Hyperledger Fabric*. 2019. URL: <https://opensource.com/article/19/9/introduction-hyperledger-fabric>.
- [55] Farzad Kiani. *Blockchain Solution for IoT Security*. Tech. rep. Istanbul, 2018.
- [56] Hong-Ning Dai, Zibin Zheng, and Yan Zhang. *Blockchain for Internet of Things: A Survey*. Tech. rep. Macau, 2019.
- [57] Christophe Jospe. *When it comes to blockchains and energy usage*. 2019. URL: <https://medium.com/nori-carbon-removal/when-it-comes-to-blockchains-and-energy-usage-dca8a76b88e>.
- [58] Jesse Morris and Sam Hartnet. *The argument for public blockchains in the energy sector*. 2019. URL: <https://www.greenbiz.com/article/argument-public-blockchains-energy-sector>.
- [59] Jon Truby. “Decarbonizing Bitcoin: Law and policy choices for reducing the energy consumption of Blockchain technologies and digital currencies”. In: *Energy Research and Social Science* 44 (1 2018), pp. 339–410.
- [60] Alex De Vries. “Bitcoin’s Growing Energy Problem”. In: *Joule* 2 (5 2018), pp. 801–805.
- [61] Kayla Matthews. *Ways to counter blockchain’s energy consumption pitfall*. 2019. URL: <https://www.greenbiz.com/article/4-ways-counter-blockchains-energy-consumption-pitfall>.
- [62] Akash Takyar. *Proof of Work vs. Proof of Stake: An In-Depth Discussion*. 2019. URL: <https://dzone.com/articles/the-proof-of-work-vs-proof-of-stake-an-in-depth-di>.
- [63] Karl J. O’Dwyer and David Malone. “Bitcoin Mining and its Energy Footprint”. In: Limerick: ISSC, 2014.
- [64] Alicia Naumoff. *Why Blockchain Needs Proof of Authority Instead of Proof of Stake*. 2017. URL: <https://cointelegraph.com/news/why-blockchain-needs-proof-of-authority-instead-of-proof-of-stake>.
- [65] Alejandro R. Pedrosa and Giovanni Pau. “ChargeItUp: On Blockchain-based technologies for Autonomous Vehicles”. In: Munich: CryBlock, 2018.
- [66] Risbah Jain and Aniket Dogra. *Solar Energy Distribution Using Blockchain and IoT Integration*. Tech. rep. Noida, 2018.
- [67] Xiaonan Wang et al. “Blockchain-based smart contract for energy demand management”. In: *Energy Procedia* 158 (2019), pp. 2719–2724.

-
- [68] Junyeon Hwang et al. “Energy Prosumer Business Model Using Blockchain System to Ensure Transparency and Safety”. In: *Energy Procedia* 141 (2017), pp. 194–198.
- [69] Jianbin Gao et al. “Grid Monitoring: Secured Sovereign Blockchain Based Monitoring on Smart Grid”. In: *IEEE Access* 6 (1 2018), pp. 9917–9925.
- [70] Zhiyi Li et al. “Blockchain for decentralized transactive energy management system in networked microgrids”. In: *The Electricity Journal* 32 (2019), pp. 58–72.
- [71] Amanda Ahl et al. “Review of blockchain-based distributed energy: Implications for institutional development”. In: *Renewable and Sustainable Energy Reviews* 107 (107 2019), pp. 200–211.
- [72] Valentino Crespi, Aram Galstyan, and Kristina Lerman. “Comparative Analysis of Top-Down and Bottom-up Methodologies for Multi-Agent System Design”. In: Utrecht: ACM, 2005.
- [73] Willemien Visser and Jean-Michel Hoc. *Expert Software Design Strategies*. Ed. by Jean-Michel Hoc, T. Green, and R. Samurcay. Academic Press, 1990, pp. 235–249.
- [74] Sabine Sonnentag, Cornelia Niessen, and Judith Volmer. *Expertise in Software Design*. Ed. by Anders K. Ericsson. Cambridge: University Press, 2006, pp. 373–387.
- [75] Gregory McFarland. “The Benefits of Bottom-up Design”. In: *ACM Sigsoft Software Engineering Notes* 11 (5 1986), pp. 43–51.
- [76] Matei Ripeanu. “Peer-to-Peer Architecture Case Study: Gnutella Network”. In: Linkoping: IEEE, 2001.
- [77] Rudiger Schollmeier. “A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications”. In: Linkoping: IEEE, 2001.
- [78] Elli Androuki et al. “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains”. In: EuroSys 2018. Porto, 2018.
- [79] Qassim Nasir et al. “Performance Analysis of Hyperledger Fabric Platforms”. In: *Security and Communication Networks* 2018 (2018).
- [80] Hyperledger. *Channels*. 2020. URL: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/channels.html>.
- [81] Vitalik Buterin. *Ethereum Whitepaper*. Tech. rep. 2013.
- [82] Xi Tong Lee et al. “Measurements, Analyses, and Insights on the Entire Ethereum Blockchain Network”. In: ACM. Taipei, 2020.
- [83] Matevz Pustisek and Andrej Kos. “Approaches to Front-End IoT Application Development for the Ethereum Blockchain”. In: *Procedia Computer Science* 129 (2018), pp. 410–419.
- [84] Ethereum. *Networks*. 2020. URL: <https://ethereum.org/en/developers/docs/networks/>.
- [85] GoQuorum. *GoQuorum Enterprise Ethereum Client*. 2020. URL: <https://docs.goquorum.consensus.net/en/stable/>.
- [86] Arati Baliga et al. *Performance Evaluation of the Quorum Blockchain Platform*. Tech. rep. Pune, 2018.
- [87] Richard Gendal Brown et al. *Corda: An Introduction*. Tech. rep. 2016.

-
- [88] Peter Robinson. *The merits of using Ethereum MainNet as a Coordination Blockchain for Ethereum Private Sidechains*. Tech. rep. 2019.
- [89] Peter Robinson. “The merits of using Ethereum MainNet as a Coordination Blockchain for Ethereum Private Sidechains”. In: *The Knowledge Engineering Review* 35 (2020).
- [90] Fatima Leal, Adriana E. Chis, and Horacio Gonzalez-Velez. “Performance Evaluation of Private Ethereum Networks”. In: *SN Computer Science* 285 (1 2020).
- [91] Markus Schaffer, Monika di Angelo, and Gernot Salzer. *Performance and Scalability of Ethereum Blockchains*. Cham: Springer, 2019, pp. 103–118.
- [92] Pieter Hartel and Mark van Staalduinen. *Truffle tests for free - Replaying Ethereum smart contracts for transparency*. Tech. rep. Singapore, 2019.
- [93] M. Vinod et al. “Theoretical and industrial studies on the electromechanical relay”. In: *International Journal Services and Operations Management* 29 (3 2018), pp. 312–331.
- [94] Russell C. Mason. *The Art Science of Protective Relaying*. 2011. URL: <https://www.gegridsolutions.com/multilin/notes/artsci/artsci.pdf>.
- [95] Konglong Tang et al. “Design and Implementation of Push Notification System Based on MQTT Protocol”. In: International Conference on Information Science and Computer Applications. 2013.
- [96] R. A. Atmoko, R. Riantini, and M. K. Hasin. “IoT real time data acquisition using MQTT protocol”. In: *Journal of Physics: Conference Series* 853 (2017).
- [97] Dipa Soni and Ashwin Makwana. “A Survey on MQTT: A Protocol of Internet of Things (IoT)”. In: International Conference On Telecommunication, Power Analysis and Computing Techniques. 2017.
- [98] Dave Bryson, Kelley Burgin, and Gloria Serrao. *Blockchain Protocol Security Analysis*. Tech. rep. Annapolis Junction, 2018.
- [99] Lucianna Kiffer et al. “Under the Hood of the Ethereum Gossip Protocol”. In: Proceedings of the Financial Cryptography and Data Security. St. George’s, 2021.
- [100] Ryan Cordell. *Ethereum Virtual Machine (EVM)*. 2020. URL: <https://ethereum.org/en/developers/docs/evm/>.
- [101] Ethereum. *Intro To Ethereum*. 2021. URL: <https://ethereum.org/en/developers/docs/intro-to-ethereum/>.
- [102] Katie Okay. *Nodes and Clients*. 2021. URL: <https://ethereum.org/en/developers/docs/nodes-and-clients/>.
- [103] Andrey Petrov. *An economic incentive for running Ethereum full nodes*. May 2018. URL: <https://medium.com/vipnode/an-economic-incentive-for-running-ethereum-full-nodes-ecc0c9ebe22>.
- [104] Thibaut Sardan. *What is a light client and why you should care?* July 2018. URL: <https://www.parity.io/what-is-a-light-client/>.
- [105] Praveen M. Dhulavvagol, Vijayakumar H. Bhajantri, and S. G. Totad. “Blockchain Ethereum Clients Performance Analysis Considering E-Voting Application”. In: *Procedia Computer Science* 167 (2020), pp. 2506–2515.

-
- [106] OpenEthereum. *OpenEthereum Documentation*. 2021. URL: <https://openethereum.github.io/index>.
- [107] Mateusz Jedrzejewski. *Welcome to Nethermind*. 2020. URL: <https://docs.nethermind.io/nethermind/>.
- [108] Edgars Nemse. *JSON-RPC vs REST for distributed platform APIs*. Apr. 2018. URL: <https://www.radixdlt.com/post/json-rpc-vs-rest/>.
- [109] Don Kiely. *The JavaScript Same-Origin Policy*. Dec. 2012. URL: <https://www.itprotoday.com/web-application-management/javascript-same-origin-policy>.
- [110] Yoichi Hirai. *Defining the Ethereum Virtual Machin for Interactive Theorem Provers*. Cham: Springer, 2017, pp. 520–535.
- [111] Felix Adler, Dennis Kitzmann, and Marc Jansen. *Analysis of Costs for Smart Contract Execution*. Ed. by Javier Prieto et al. Springer, 2020, pp. 153–156.
- [112] Dannen. Chris. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. 1st. New York: Apress, 2017.
- [113] Ganache. *Ganache Overview*. 2020. URL: <https://www.trufflesuite.com/docs/ganache/overview>.
- [114] James M. Fiore. *Conventional Current Flow and Electron Flow*. Mar. 2021. URL: [https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Electronics/Book%3A_DC_Electrical_Circuit_Analysis_-_A_Practical_Approach_\(Fiore\)/03%3ASeries_Resistive_Circuits/3.02%3A_Conventional_Current_Flow_and_Electron_Flow](https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Electronics/Book%3A_DC_Electrical_Circuit_Analysis_-_A_Practical_Approach_(Fiore)/03%3ASeries_Resistive_Circuits/3.02%3A_Conventional_Current_Flow_and_Electron_Flow).
- [115] Espressif. *ESP8266EX Datasheet*. Oct. 2020. URL: https://www.espressif.com/en/support/documents/technical-documents?keys=&field_type_tid%5B%5D=14.
- [116] Saburo Muroga. *Ultra Large-Scale Integration Design*. Academic Press, 2001, pp. 245–267.
- [117] Sonoff. *POWR2 Smart Switch*. URL: <https://sonoff.tech/product/diy-smart-switch/powr2/>.
- [118] Theo Arends. *Tasmota*. 2020. URL: <https://tasmota.github.io/docs/About/>.
- [119] Jignesh Parmar. *How reactive power is helpful to maintain a system healthy*. 2011. URL: <https://electrical-engineering-portal.com/how-reactive-power-is-helpful-to-maintain-a-system-healthy>.
- [120] Michel Malengret. *Definition of Apparent Power in 3-Phase 4-Wire Non-Sinusoidal Power Systems*. Tech. rep. Cape Town, 2008.
- [121] Steve Winder. *Power Supplies for LED Driving*. 2nd. Newnes, 2017.
- [122] Nick Heath. *How IBM’s Node-RED is hacking together the internet of things*. Mar. 2014. URL: <https://www.techrepublic.com/article/node-red/>.
- [123] Lucy Hattersley. *Raspberry Pi 3B+ Specs and Benchmarks*. URL: <https://magpi.raspberrypi.org/articles/raspberry-pi-3bplus-specs-benchmarks>.
- [124] Sam Richards. *Nodes and Clients*. Apr. 2021. URL: <https://ethereum.org/en/developers/docs/nodes-and-clients/>.
