Blockchain Facilitated Energy Exchange for Open-Grid Applications

Christopher Maree - 110946

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: This paper presents a mechanism to facilitate energy exchange between producers and consumers, utilizing a decentralised blockchain based platform, enabling a trustless mechanism which would obviate the need for a third party. Applicability within a South African and microgrid context is explored and the validity of the solutions is critically analyzed. A real world implementation of the technology is then presented with the construction of five Raspberry Pi based smart meters connected to the Ethereum blockchain interacting with custom smart contracts. The proposed solution is then critically analysed and requirements to enable the technology to scale to a usable level are then proposed.

Key words: Blockchain, Decentralised, Energy Exchange, Microgrids, Smart Meters

1. INTRODUCTION

The Effective Energy Transitions Index 2018 done by the World Economic Forum placed South Africa at 113th out of 114 countries in a study investigating a countries ability to balance energy security and access, while considering environmental sustainability and affordability[1]. Coupled with the fact than more than 90% of South Africa's energy is generated through coal power plants and that it is currently very difficult to sell energy back to the grid, the South African energy market is ripe for radical transformation in how energy is produced, transported and consumed[1]. Distributed energy production, through the use of renewable energy, will play a pivotal role in this transformation.

The future South African energy market should involve a transition from a pure producer/consumer model to a more decentralised hybrid model, involving multiple producers. Consumers are becoming the new producers as the energy market becomes more democratized with renewable energy becoming more accessible than ever before[1].

Globally, there are 1.2 billion people living in off-grid communities, over half of which are located in Sub-Saharan Africa[2]. Microgrids will play an integral role in enabling energy access to these communities where connection to the national electricity grids is not viable.

Given the inevitable, radical transformation of the energy sector within South Africa and the need for energy access globally, an equally important part of this transition will be the underlying accounting mechanism used to facilitate energy exchange between producers and consumers. This paper presents a novel mechanism offering the ability for producers and consumers to meet in an equally beneficial environment that can facilitate secure, decentralized, energy exchange through the use of blockchain technology. This mechanism unlocks the power of renewable energies by allowing the formation of self sustaining microgrids in isolated networks as well as providing a scalable way to augment the current energy production systems within South Africa.

This paper consists of three main sections: firstly, background information to identify the need for distributed power production and technology contextualization to provide the framework upon which the implementation is based. Secondly, the implementation details of the proposed solution are discussed, along with the testing setup. Lastly, the implementation is critically evaluated as well as the viability of blockchain technology as a whole in the energy sector.

2. LITRERATURE REVIEW

There are a number of projects that attempt to address energy distribution using blockchain technology.

Reference [3] uses blockchain as an accounting mechanism for the resale of energy. They do not provide any ability to sell energy back to the grid but rather act as a traditional energy provider using blockchain as their billing mechanism. Their white paper provides useful insight into the tokenization of energy and the interconnection with existing infrastructure.

Reference [4] uses blockchain as a funding mechanism to raise capital by selling future energy production upfront in the form of tradable smart contracts. Reference [4]'s insight into token economics proves very useful in linking blockchain to energy economies.

References [5, 6] utilize blockchain as the underlying accounting mechanism in the creation of microgrids, such as those formulated in Brooklyn and South Australia. They provide invaluable insight into the complexities of integrating blockchain within the energy sector.

Additionally, there is a myriad of research into the applicability of microgrids within Sub-Saharan Africa, such as References [2, 7]. These papers provide the justification for the need for microgrids as well as the problems with current accounting mechanisms.

3. BACKGROUND

There are a number of factors that add to the justification for microgrids and distributed energy production.

3.1 South African Legislation

It is currently difficult to sell energy back to the grid within South Africa due to legislation imposed by the government[8]. However, this legislation is currently being contested by the City of Tshwane's flagship policy on Embedded Power Generation (EPG), which aims to promote small-scale solar power generation by residents[9]. While this updated legislation is currently not in effect, the energy generation climate within South Africa is rapidly changing towards an environment that will facilitate the sale of energy back to the grid by consumers. As a result, in the near future, it will be possible to sell energy back to the grid, creating incentives for the installation of renewable energy sources.

3.2 Centralized Energy Production Problem

South Africa's energy production environment hinges on once central utility: Eskom. They are responsible for the production and distribution of the vast majority of electricity within South Africa. As a result, heavy energy losses are incurred due to the need to transport energy over long distances to provide electricity to consumers that are far away from production facilities[10]. This problem becomes even worse if you consider isolated communities within Sub-Saharan Africa that currently do not have any access to electricity. The cost of installing power distribution infrastructure is prohibitive and the losses would be very high[2].

Distributed energy production would drastically reduce the cost of energy transportation as the electricity production can be physically closer to the point of consumption.

3.3 Energy Distribution Accounting Problem

Electricity production utilities require a mechanism to correctly bill users based on their energy consumption. There are currently two ways to purchase electricity in South Africa from the national utility: 1) Pay-as-you-go model that has inherent inaccuracies due to the infrequency of measurements. 2) Pre-paid model that addresses these inaccuracies by allowing the user to load energy tokens and then billing for every kWh consumed.

However, neither of these models facilitate the ability to sell energy back to the national grid. These models do not provide additional incentives for the installation of renewable energy systems within the household as there are no mechanisms to profit from surplus energy produced. Additionally, the user has to trust the accounting mechanisms used by the national utility, including the manner through which energy consumption is recorded and the associated billing thereof. These restrictions inherently mean that there is no platform upon which a microgrid can be created, wherein energy exchange occurs directly between producers and consumers.

In the context of isolated microgrids, the accounting problem becomes even more apparent as there is no central utility to facilitate the billing process. This means that microgrids would need to either go without a billing mechanism, relying on face to face communication, which does scale, or they would need to introduce a third party to act as the accounting provider, such as banks or mobile network operators[2].

4. PROPOSED SOLUTION

This paper presents meterBlock; a decentralised blockchain facilitated energy exchange platform that aims to address the problems outlined with the current electricity environment within South Africa and isolated microgrids within Sub-Saharan Africa. meterBlock allows for the direct exchange of energy between producers and consumers, without the need for any middlemen. Any number of producers and consumers can join the platform and begin exchanging energy with each other. Users have assured security in the underlying accounting mechanism, due to the trustless, distributed nature of blockchain technology. Through the use of smart meters, utilizing per second billing, high measurement accuracy can be guaranteed resulting in a fair billing system for both producers and consumers alike.

5. TECHNOLOGICAL CONTEXTUALIZATION

A brief contextual understanding of blockchain technology is required to understand the implementation.

5.1 Blockchain

A blockchain is a public, peer-to-peer, digital ledger, used to record transactions chronologically. It represents an immutable, persistent record that can never be altered or changed by anyone due to the distributed nature of the consensus algorithm. Miners, all around the world, offer up their computation power to verify transactions (in "blocks") to secure the network in exchange for a reward. This is known as Proof of Work consensus[11]. It is this distribution of computational power, used in the verification of blocks, that results in the inherent security of the network; if one wants to alter the ledger, one would require to have more than half of the total computational power on the network, commonly known as a 51% attack[11]. Note that there are other consensus algorithms, such as Proof of Stake, but the underlying immutability concepts remain the same.

5.2 Ethereum, Smart Contracts and Consortiums

meterBlock is built upon the Ethereum blockchain. For a full rational into the selection of Ethereum over other blockchains, see Appendix A1. Ethereum aims to create the ultimate protocol for building decentralised applications. Ethereum implements a Turing-complete programing language built into its blockchain, enabling anyone to create and deploy smart contracts[12].

Ethereum can be thought of as a decentralised, world computer, disrupting the current client-server model[12]. Smart contracts are compiled to a low-level bytecode language that is then run within the Ethereum Virtual Machine(EVM). For the purposes of this paper, in an attempt to keep the conceptual elements of blockchain simple, smart contracts can be thought of as autonomous programs that execute automatically when specific predefined conditions are met[13].

A consortium is private blockchain that is operated between a group of trusted nodes. As a result, custom consensus rules can be implemented that result in different properties, such as "free" transactions and faster processing time.

5.3 Blockchain Technology in the Energy Sector

Blockchain systems provide the facility for a peer-to-peer exchange of assets without requiring the introduction of a trusted third party to facilitate the exchange. This application is very apparent in the energy sector as the removal of the middle man in the exchange between producers and consumers reduces cost for all parties involved. Additionally, the removal of a third party makes the energy exchange safer as no trust needs to be placed in the hands of an external organisation.

6. IMPLEMENTATION DETAIL

meterBlock's solution has three main layers to the design, each of which will be explored to provide a holistic overview of the proposed solution. These sections are namely: 1) custom Ethereum-enabled smart meters; 2) Ethereum smart contracts and distributed databases; 3) a web3-enabled user interface. An overview of how these layers interact can be seen in diagram 1 with a full system technology architecture stack available in Appendix A2. All software stacks discussed below can be seen here on github.

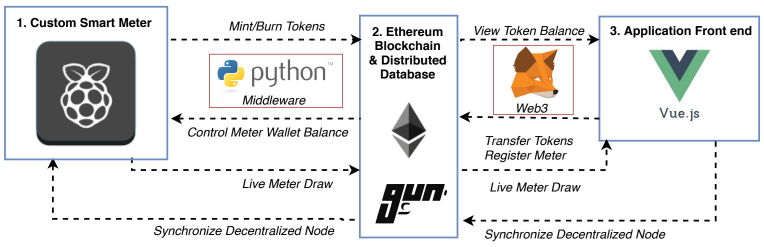

1. Custom Smart Meter
Mint/Burn Tokens
python
Middleware
Control Meter Wallet Balance
Live Meter Draw
Synchronize Decentralized Node
2. Ethereum Blockchain & Distributed Database
gun
View Token Balance
Web3
Transfer Tokens Register Meter
Live Meter Draw
Synchronize Decentralized Node
3. Application Front end
Vue.js

Figure 1 : Highlevel system overview showing interconnection between the custom meters, distributed technologies and application frontend

6.1 *Custom Smart Meters*

Physical hardware is required to be installed within each household of the grid to measure and record consumption and production of electricity. Unlike most modern South African smart meters, meterBlock's meters require the ability to accurately identify the direction of energy flow to correctly credit producers and bill consumers.

Raspberry Pis were used in the construction of meterBlock's smart meters to provide a framework to interconnect with the hardware required to measure load draw, control load state and interact with the blockchain. More simplistic microcontrollers could be used in the load controlling and measurement components but they do not provide adequate resources to connect to the Ethereum blockchain[14].

6.1.1 Load Controlling Circuitry meterBlock's meters require the ability to control the state of a large load, such as a house. Most South African household's mains breakers are ≈75A rated[1]. Therefore, to control such a large load, a 75A contactor was used. This rating also conforms to the SANS 1524 current rating of a value between 60A and 80A for pre-paid meters within South Africa[15]. The contactor requires 220V to drive the electromagnetic coil within it, so a relay was used to switch the controlling signal to the contactor. To power the relay, a Bipolar Jjunction Transistor (BJT) circuit was constructed to to enable the 3.3V General Purpose Input/Output (GPIO) pins from the Raspberry Pi to control the 5V relay. The diagram in Appendix A3 depicts this circuit. The GPIO pins on the Raspberry Pi were controlled using `gpiozero`; a simple interface to GPIO devices with Raspberry Pi[2].

6.1.2 Measurement Circuitry Each household's meter needs to be able to accurately record the exact electricity consumption of a house as well as the direction of energy flow. To accomplish this, a combination of voltage and current sensors were implemented to measure the instantaneous power consumed by the house. Raspberry Pi's do not have a built in analog-to-digital converter (ADC), so an external MCP3202 12-bit, 2 channel ADC was used in conjunction with the Raspberry Pi's Serial Peripheral Interface (SPI).

Assuming household loads are never purely capacitive or inductive, the phase offset between voltage and current can be used to determine the direction of energy flow. If energy is flowing into the meter, then voltage and current will be directly inphase with one another. If, however, energy is flowing in the opposite direction, then the measured voltage and current will appear to be π out of phase with one another. Appendix A4 shows the measurement circuit as well as inphase and out of phase measurements conducted with the meterBlock measurement meter on a 10A RMS load.

By taking the sum of products over a number of high frequency samples, encompassing a number of AC waveforms, the direction of energy flow will correspond to the polarity of the sum of products. Equation 1 below represents this method for defining energy flow polarity over k samples.

$$\begin{aligned} if \sum_{n=0}^{k} v_n i_n > 0 \rightarrow consuming \\ if \sum_{n=0}^{k} v_n i_n < 0 \rightarrow producing \end{aligned} \tag{1}$$

6.1.3 Python Middleware Scripts Python was chosen as the language of choice to run on the Raspberry Pi's as it provides an easy interface to interconnect all layers, from the load controller all the way up to the blockchain. Separation of concerns was implemented in all layers of the software design used on the Raspberry Pi's, enabling easy upgradability through modularity. A full diagram outlining the interconnection of different key elements as well as an explanation for each script can be seen in Appendix A5.

6.2 *Ethereum Blockchain and Distributed Databases*

A number of Ethereum smart contracts were created in `Solidity` as the underlying accounting mechanism used for the allocation and deallocation of tokens within the meterBlock ecosystem. Each household meter acts as a node within the blockchain and has its own Ethereum wallet with a corresponding private key stored on the meter. This configuration enables the meter to interact with the blockchain and smart contracts.

`GunDB` - a distributed, peer-to-peer, graph database - was used to provide live usage metrics for the meters. Users require the ability to see the current and historic draw of their meters. Rather than congesting the blockchain with this information, an alternative, while still decentralized, mechanism was used to provide these metrics.

6.2.1 Token Economics The underlying logic that defines how energy is traded between producers and consumers is fundamental to the applicability of the meterBlock system. Three different energy trading configurations were identified and implemented. A full analysis of these can be found in Appendix A6. After detailed analysis, it was decided that the simplest and most applicable model was the tokanisation of the kWh, known as a "*KraG token*". This means that each token within the ecosystem corresponds to one unit of energy equal to a kWh. Upon the creation of energy (a producer exporting to the grid) a new token is minted. In other words, new tokens enter circulation in conjunction with the production of energy. Upon the consumption of energy (a consumer drawing energy from the grid), tokens are burnt. In other words, tokens are removed from circulation in conjunction with the consumption of energy.

6.2.2 Leveraging Existing Cryptocurrency Asset Exchanges The underlying economics that define the price that energy is traded at is extremely complex. There are multiple factors that must be taken into account in order to correctly price such a diverse, scarce asset. By tokenizing the kWh and enabling producers to mint their own tokens and then sell them on public cryptocurrency asset exchanges, normal economic forces can be leveraged in the process of price discovery. The supply and demand of KraG tokens, as traded on public open exchanges will be the defining factor that corresponds to the energy price.

6.2.3 Token Standardization The tokens implemented conform to the 20th Ethereum Request for Comments (ERC20) token standard[12]. This means that the tokens implemented within meterBlock are able to be traded on any current cryptocurrency asset exchange (there are currently more than 500 cryptocurrency exchanges worldwide at the time of writing[13]). The tokens can be transferred between wallets, loaded into meters and withdrawn from meters.

6.2.4 Live Pricing Tariffs Electrical energy varies in value at different times of the day. For example, under high grid loads, the value of energy increases as there is an increased demand. To address this, live pricing can be implemented to facilitate per second billing with dynamic pricing tariffs. To implement this on Ethereum a pricing oracle was experimented with to show that it can be successfully implemented. An oracle is a mechanism for providing offchain information (such as the current network load and resultant pricing information) to an on chain entity, such as a smart contract. This can be conceptualised as a black box that sits off the blockchain and is responsible for informing the meters of the current energy price. This price is then used in the billing/crediting process for consumers/producers.

6.2.5 Meter Enrollment All meters within the meterBlock ecosystem are required to be enrolled in order to verify that the meter's public Ethereum address does indeed correspond to a physical meter commissioned and approved by a verified third party. This is vital as each meter has the ability to mint its own tokens and as such this function needs to be restricted via a smart contract to only valid meters.

6.2.6 Smart Contract Design The ERC20 smart contracts created for meterBlock were based off the standards specified by OpenZeppelin. OpenZeppelin is a library for writing secure smart contracts on Ethereum by using battle-tested frameworks. Their contract libraries secure 4.5 billion dollars worth of digital assets so there is some degree of surety that the underlying code base is secure. The contracts implemented require additional functionality that the OpenZepplin standards do not provide, such as the ability to register meters and custom minting/burning functionality on the production/consumption of electricity. For this, a smart contract was created to facilitate the additional features. A full description of the custom smart contract and associated inheritance diagrams can be seen in Appendix A7

6.2.7 Smart Contract Testing Extensive unit testing was performed on the smart contracts to ensure that they performed as expected. A total of 64 unit tests were written in javascript and tested using the `Truffle` framework to verify the security and operation of the implemented smart contracts. Additionally, the contracts were tested using `Mythril`; a static analysis tool used to identify known contract vulnerabilities.

6.2.8 System Scalability Considerations The proposed system can be configured to communicate with any implemented Ethereum blockchain. This could be the main public Ethereum Network to have maximum system security and immutability[16]. Alternatively, this could be a private, consortium-like blockchain, hosted between the meters themselves[16]. The problem is that the main network can only handle ≈ 15 transactions per second at present, making scalability a major concern. To deal with this, the meters only communicate with the blockchain once every week, drastically reducing the total draw on the network. If a consortium is used, this network speed consideration is no longer a problem.

6.2.9 Live Metrics via Distributed Database Users of the platform will want to know what their current draw and token balance is at a particular point in time, not what it was the last time the meter communicated with the blockchain. To facilitate this, GunDB was used to provide real time usage metrics. Each meterBlock meter stores information relevant to their particular draw and updates the database every two seconds. When a user requests information on their meter, the relevant information is shared with their browser upon which the database is duplicated. The GunDB implementation details and associated data structure can be found in Appendix A8.

6.3 User Frontend

A user facing frontend was created to facilitate interaction with smart meters. This frontend provides the administrator with the ability to enroll new meters and view the current health of the energy grid as a whole. The frontend enables a normal user to view their current token balance, transfer tokens between their personal wallets and the meter's wallet and view their current (and historic) household draw. Screenshots of the frontend can be seen in Appendix A9.

As with all other layers of the technology stack, the frontend embodies the concept of separation of concerns wherein the frontend logic has been separated away from all underlying implementation logic. Utility scripts were created to enable the frontend components to query the blockchain and distributed databases without the need to interact directly with them. This separation enables simple upgradability and maintainability. For a full breakdown of the frontend technology stack, see Appendix A9.1

7. IMPLEMENTATION EXPERIMENTAION

A number of custom smart meters were created to test the accuracy and usability of the system.

7.1 Measurement Accuracy

The custom meter measurement circuitry proved to be accurate within 3V of the actual reading, when tested on 230V mains voltage against a multimeter. Current readings were also consistently accurate within 200mA of actual readings. Current tests were conducted up to a 15A load and showed linearity through the range. SANS 1524 specifies that meters must be a class index 2 or better to be used within South Africa, resulting in an accuracy of 2% from the measured amount[15]. The custom meter created falls well within these requirements.

7.1.1 Floating at zero draw The meter would report low current readings when there was no load attached as a result of floating pins and associated measurement noise. To combat this effect, the `loadController.py` measurement code was modified to only show any form of draw if the load was above 50W, a value above the noise.

7.2 *Microgrid Simulation*

A total of 5 smart meters were constructed to conduct a full system test to simulate a microgrid. The test microgrid configuration can be found in Appendix A10. One meter was placed in a pure producer configuration, only generating energy for the rest of the microgrid. Three other meters were placed in pure consumer mode, consuming energy from the pure producer. Lastly, a hybrid was constructed that could switch between producer and consumer mode to simulate a house either pulling energy from the grid or contributing back to the grid. This hybrid simulated a house with solar panels going through the day night cycle of having surplus energy during the day and a deficit at night. Four different loads were connected to the three pure consumers and the hybrid respectively.

Ganache was used to host a private Ethereum blockchain for the testing environment[1]. Ganache's default unlocked, funded accounts were used so that each node would not need to have Ether transferred to it to pay for transaction fees. Clearly, this configuration is only applicable for the sake of a proof of concept.

The system behaved as expected, with correct minting and burning of tokens on production and consumption. The measurement and reporting of load draws was correctly displayed on the user frontend. The distributed database performed adequately, reflecting updates in real time.

8. SYSTEM LIMITATIONS AND CRITICAL ANALYSIS

The proposed system operates as intended under the small scale experimentation of five nodes, within a simulated microgrid context. There are, however, a number of major problems in a few areas that arise as soon as you try to scale the system up to any realistic number of users.

8.1 *Meter Hardware Limitations*

The custom meters have a few limitations, namely:

8.1.1 Need to Trust Meters The network needs to trust that no meters within it have been compromised. Someone can simply bypass their meter and get free energy, or, in a worse case scenario, flip the direction of their meter around resulting in the consumption of energy crediting them as a producer. This problem is not unique to meterBlock and is apparent with current smart meters used today. The risk from this vulnerability can be minimized by making the meters physically difficult to tamper with, by placing them in hard to reach locations, or by including tamper proof seals that disable the meter if the housing is opened.

8.1.2 Deployability and Maintainability The testing configuration, as outlined before, requires a high degree of manual setup and configuration to perform tests. Updating software on each node requires manual intervention. This is by no means scalable or maintianable with any realistic number of nodes. A solution to this problem would be to containerize the meter's software into a reproducible microservice through a platform like `Docker`. Then, over the air updates can be distributed to the meters using automatic docker management utilities, such as `Resin` that provide the ability to deploy and configure docker containers on IoT platforms.

8.1.3 Connectivity Requirement If the main Ethereum network is to be used, then each meter requires to connect to the internet every week to update the blockchain state. This is not realistic in many situations where internet connectivity is not stable or present at all. A consortium could be used wherein the meters communicate on a private network established between the meters themselves, not requiring any communication to the internet. Alternatively, a GSM module could be used to provide connectivity.

8.1.4 Cost of Physical Hardware The Raspberry Pis used in the design are not the cheapest way to interface with the hardware and the blockchain. Future iterations of this project should involve the use of cheaper hardware.

8.2 *Blockchain Limitations*

The proposed blockchain solution is limited in a few ways:

8.2.1 Scalability The proposed design involves the meters communicating with the blockchain once a week to reduce blockchain network traffic. If the proposed system was implemented in the real world, with each node communicating with the main Ethereum blockchain once per week, only 9.5 million households would be able to successfully interact with the system before the meters overwhelmed the total network traffic for the main Ethereum network at a cap of $\approx$15 transactions per second[13]. This poses a huge scalability issue and does not even consider the network costs associated with these transactions.

There are a number of possible scaling improvements that could address this fundamental bottleneck, including both layer 1 (on-chain) and layer 2 (off-chain) scaling solutions. The three key scaling solutions of relevance to meterBlocks viability are in the form of Sharding (on-chain) State Channels and Plazma (both off-chain).

Sharding involves spliting the entire state of the network into partitions, called shards, so that each contain their own independent component of the network state and transaction ledger. Shards can then be further broken down into subshards, resulting in several levels of nodes that can exist[17]. This splitting process means that not all nodes within the blockchain are required to process and store every transaction, which results in optimization of the whole network. Sharding could, theoretically, take the network speed into the hundreds of thousands or even millions of transactions per second[16].

State channels leverage the fact that not every payment needs to be agreed upon by the global blockchain to be deemed valid. You only need the participating parties to agree to the transaction and have valid proof that assets were transferred between them. This is directly relevant in the context of energy exchange where a payment channel can be opened up between a producer and consumer, multiple trades are executed over an extended period of time and the channel is closed later on[18]. This can be thought of as opening a tab at a bar and only needing to pay at the end of the evening.

Plazma offers the ability to create child blockchains that are attached to the main Ethereum blockchain (called side chains). Further levels of children chains can be created within the Plazma side chains offering even further scalability. Application specific side chains enable users to directly interact with the side chain and as such scalability gains can be exponential[19].

Another possible solution to the scaling problem is the construction of a consortium between the meters within the community. A consortium can be seen as a mini, private blockchain where one can define their own consensus protocols and rules. One would need to think carefully about the underlying consensus protocol used within the consortium as this would define its applicability. Proof of work is clearly not an option here due to the high energy requirements of the protocol. Proof of stake could pose a possible solution but this opens the network up to being controlled by a cartel of energy producers who hold the majority of tokens, and, as a result, can influence and sensor the blockchain.

The key takeaway from these scaling solutions is that it is not an insurmountable problem, but the technology is not yet ready for production usage. Ethereum and other blockchain technology still need a few years to mature until an application of this magnitude could operate at any kind of scale.

8.2.2 Paying Gas Fees All Ethereum transactions require the user to spend an amount of Ether (the base currency) to pay for transaction fees. In the context of meterBlock, this means that every time the meters connect to the main Ethereum network, they require to spend Ether to facilitate these transactions. As a result, each meter will required to be loaded with a set amount of Ether to pay for the transactions into the future. Over scale, this becomes a non-trivial amount of money that needs to be considered. Additionally, if the meters run out of Ether, they will no longer be able to pay the gas prices and so will stop functioning. A possible solution to this links into the scalability solutions outlined before wherein transactions on the mainnet become cheap enough that the fees become negligible. Alternatively, the use of a Consortium would address this problem.

8.2.3 Security Vulnerabilities Only meters that have been registered by the meter manufacturer are capable of minting new tokens. This restricts abuse of the mint function by non-meter accounts. However, if someone is able to compromise the security of the meter, they can in theory mint as many tokens as they want. A possible workaround to this problem is to set hard minting limits based on production capabilities of a facility when a meter is registered by the meter manufacturer. As a result, a compromised meter will only be able to mint the maximum number of tokens that they could genuinely produce in a period of time. This does however have the downside that each upgrade to someone's production facility requires an update to the smart contract from the meter's manufacturer.

An additional security consideration is that the meters themselves store their own private keys. These keys need to be stored in a very secure configuration preventing a malicious actor from stealing tokens from the meters as well as the Ether on them used to pay gas prices.

8.2.4 Token Economics Black Start The notion of minting new tokens on energy production and burning tokens on energy consumption has a fundamental problem when the system starts from black. If no one has tokens to buy energy from a producer, then the producer can't mint new tokens as no energy is leaving their production facility. As a result, no new tokens enter the market and there is a resultant liquidity problem. A possible solution to this could be to sell meters with a set amount of pre-loaded KraG tokens on them to "soft start" the economy. Alternatively, a pre-sale event could be used to generate the required liquidity.

8.2.5 Blockchain Adoption meterBlock requires that all users interact with the blockchain via a web3 enabled browser. For the purposes of this project, `Metamask` was used. This restriction means that every user that interacts with the platform will need custom software installed on their computer. This requirement could pose adoption problems until blockchain technology is more commonly adopted. A solution to this problem could be to provide a traditional web2 login interface for non-web3 ready users.

9. CONCLUSION

meterBlock proves to be a successful proof of principle, achieving the notion of decentralized energy exchange.

Microgrids and distributed production enable the sale of energy between producers and consumers and back to the national grid thereby unlocking the power of renewable energy. This will revolutionise how energy is produced, distributed and consumed.

Blockchain technology has much promise to provide a mechanism enabling energy exchange, providing a framework upon which future micro and smart grids can be built. It provides a novel way to remove the middleman in energy exchange, facilitating peer to peer trade, while maintaining security and scalability. Despite the clear advantages, the technology is still very new and requires much development until it is ready for mainstream adoption.

REFERENCES

[1] K. Schwab. "The Global Competitiveness Report 20172018." Tech. rep., Worl Economic Forum, 2017. URL Link.

[2] E. Francks. "The Future of Solar Microgrids in Sub-Saharan Africa: How social enterprises can accelerate the spread of renewable energy to off-grid communities." Tech. rep., Global Social Benefit Institute, 2017. URL Link.

[3] Gridplus. "Grid+ White paper." 2017. URL Link.

[4] WePower. "Energy financing and trading platform powered by blockchain technology.", 2018. URL Link.

[5] Electron. "Blockchain Systems for The Energy Sector.", 2018. URL Link.

[6] L03 Energy. "Blockchain based innovations to revolutionize how energy can be generated, stored, bought, sold and used, all at the local level.", 2018. URL Link.

[7] T. Reber and S. Booth. "TARIFF CONSIDERATIONS FOR MICRO-GRIDS IN SUB-SAHARAN AFRICA." *US Aid*, 2018. URL Link.

[8] Eskom. "Selling energy to Eskom." 2014. URL Link.

[9] Stephens Notoane. "Proposed Embedded Generation Policy to support and Formalise production For installations within the city." 2017. URL Link.

[10] A. V. Ketelhodt. "The impact of electricity crises on the consumption behaviour of small and medium enterprises." Tech. rep., 2008. URL Link.

[11] Bitcoin Wiki. "Proof of work.", 2016. URL Link.

[12] V. Buterin. "Ethereum en route to a million transactions per second Brave New Coin.", 2018. URL Link.

[13] CoinDesk. "How Will Ethereum Scale?", 2018. URL Link.

[14] S. Huh, S. Cho, and S. Kim. "Managing IoT Devices using Blockchain Platform." Tech. rep. URL Link.

[15] J. Kennedy. "PARTICULAR REQUIREMENTS FOR PREPAYMENT METERS." Tech. rep., 2015. URL Link.

[16] Vitalik Buterin. "Ethereum White Paper.", 2018. URL Link.

[17] Ethereum Wiki. "Sharding FAQs." URL Link.

[18] J. Coleman, L. Horne, and L. L. Xuanji. "Counterfactual: Generalized State Channels." Tech. rep., 2018. URL Link.

[19] J. Poon and V. Buterin. "Plasma: Scalable Autonomous Smart Contracts." Tech. rep., 2017. URL Link.

Appendices

Appendices relevant to the report can be found below. Afterwards, other documentation requested by the CBO can be found. This includes a reflection on group work in Appendix B, the project specification document in Appendix C and the project plan in Appendix D. Minutes taken during meetings as well as a two videos overviewing the technology implemented were submitted as project supporting material.

A Report Appendix

A1 Selection of Ethereum as Blockchain of Choise

There are a number of blockchain platforms that could potentially be used in the meterBlock design. Each of the common blockchain platforms are considered below and compared to provide a full rational into why Ethereum was chosen.

A1.1 Bitcoin is the most popular and well known blockchain. It does however have very limited scripting abilities and does not offer a Turing complete programing language capable of providing dynamic, autonomous execution of contract code. It's functionality is limited to the transfer of value from one party to another. As a result, if the Bitcoin blockchain was used, heavy modification to the underlying codebase would be required. Moreover, Bitcoin can only facilitate three transactions per second, has high transaction fees and extremely slow transaction processing times (up to twelve hours to process a single transaction). As such, payment channels would also be required to be implemented if Bitcoin was to be used. Lastly, Bitcoin uses Proof-of-work as its consensus algorithm making it inherently inefficient at reaching consensus. This ideology is diametrically opposed to that of meterBlock.

A1.2 IOTA is designed to be more scalable than traditional blockchains, with no theoretical maximum throughput. Additionally, it has no miner fees as there are no miners within the IOTA network. Rather, each node within the network performs the action that a traditional miner would do when processing a transaction. IOTA does not have blocks and as a result, no chain. It is a stream of interlinked transactions that are distributed and stored across the network through a data structure called a Tangle, a form of a DAG. Despite the apparent advantages of IOTA, it does not offer any platform to create smart contracts and as such would require the use of extensive modification to accommodate any form of complex token exchange settlements. As meterBlock aims to be a proof of principal, IOTA is not considered appropriate due to the implementation complexities associated with using it as the underlying blockchain.

A1.3 Cardano is a distributed blockchain computing platform. It uses proof-of-stake, making it inherently more efficient than any other proof-of-work alternative . Cardano does plan to offer smart contracts in the future but in its current implementation it does not have this functionality and so it is not applicable for meterBlock.

A1.4 NEO formally known as Antshares, is seen as the Chinese version of Ethereum by many. Fundamentally, it offers much of the same functionality as Ethereum with full smart contract support. Moreover, NEO's smart contract platform is language agnostic meaning that any turing complete programming language can compile down to NEO's smart contract bytecode. For example, one can write C++, C# or Javascript smart contracts that are then deployed to the NEO blockchain. NEO uses delegated byzantine fault tolerance (dBFT) as it's consensus algorithm. This makes it efficient and fast but this results in the consensus becoming more centralised than other platforms and as such makes it more susceptible to censorship. Additionally, NEO has a limited English speaking developer base making the interaction with the documentation cumbersome at times. Lastly, NEO's platform is still under development and is not considered production ready.

A1.5 Ethereum is a blockchain platform offering turing complete smart contracts. It has the biggest developer mindshare by an order of magnitude. Ethereum is currently used in more smart contract related projects than any other blockchain, with 96 out of the top 100 token based applications running on the Ethereum blockchain. Ethereum is currently using Proof-Of-Work, but will soon move to Proof-Of-Stake under the Casper update making it far more scalable and efficient. Ethereum offers the ability to easily run a consortium blockchain between trusted nodes, making the setup and operation of a private blockchain advantageous for testing and development. Lastly, Ethereum has extensive documentation and is currently considered production ready.

Due to the simplicity of implementation, ability to write Turing-complete smart contracts future sustainability due to developer mind share, true decentralization and current implemented token standards, Ethereum is chosen as the blockchain of choice for meterBlock.

It is important to note that ideas discussed in this paper can be applied to any Distributed Ledger Technology(DLT). This does not have to be in the form of Ethereum.

A2 System Technology Stack Architecture

Figure 2 shows the inter connection of the three main components within the meterBlock ecosystem, naimly the frontend, the distributed technology stack(blockchain and distributed, peer to peer database) and the custom meters. All source code associated with the implementation is available on Github, here.

Starting at the top of the stack, respective source code can be found on Github in the following locations:

- Frontend source code can be found here.
- Smart contracts and unit tests can be found here and here respectively
- Distributed graph database implementation on python can be found here here, the interconnection to node and express found here and the connection to the front end found here
- Python middleware running on each Raspberry Pi can be found here
- Code used to control the load state and interact with the measurement hardware can be found here

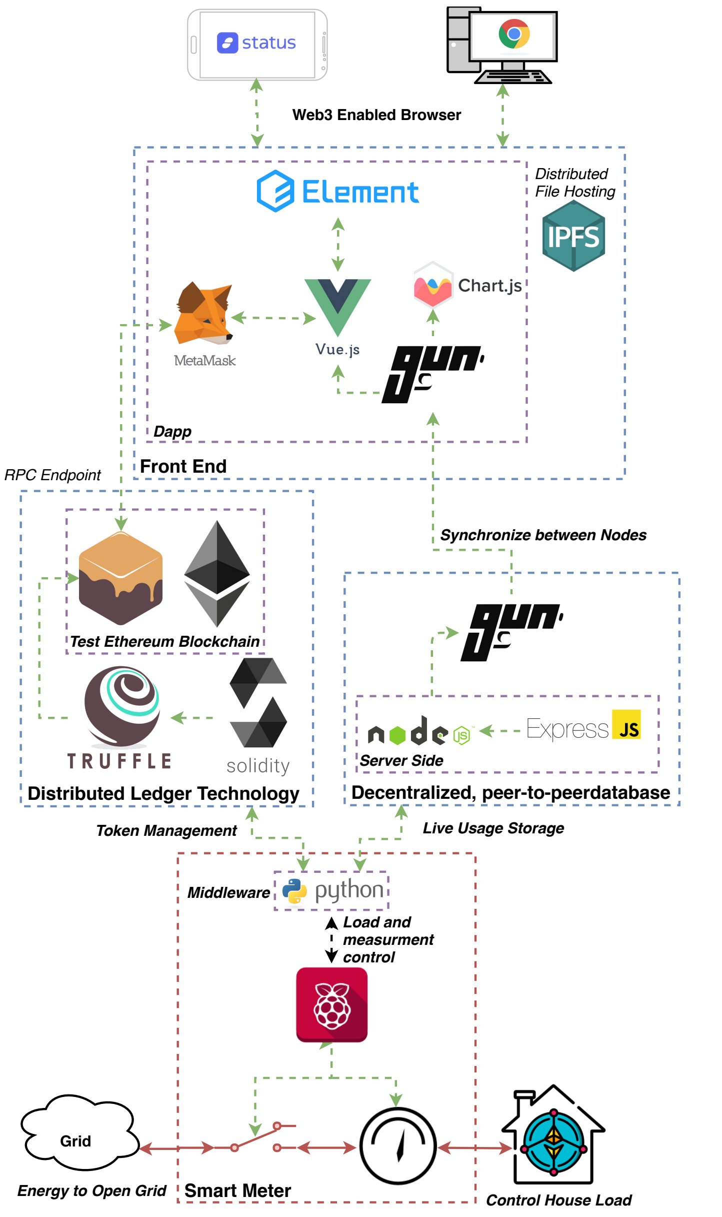


Figure 2 : Overview of all technology used within The meterBlock design, showing the interconnection between all layers

A3 House Hold Load Controller

Figure 3 outlines the controller circuitry used to toggle the state of the household load. This circuit involves driving a 75A contractor using a Raspberry Pi through the use of a relay and relay driving circuit using a BJT. Circuit simulations of the controller circuit can be found here.

The code to control this load is embedded within the `loadController.py`, utilizing `gpiozero`. Full source code for the controller can be found here.

A4 Measurement Circuity

Figure 4 show the implementation of the load measurement circuity used to measure voltage and current associated with the household load

A number of voltage and current waveforms were recorded to show the performance of the circuit. Figure 5 shows in phase measurement, corresponding to consumption of energy. Figure 6 corresponds to out of phase measurements, relating to the production of energy.

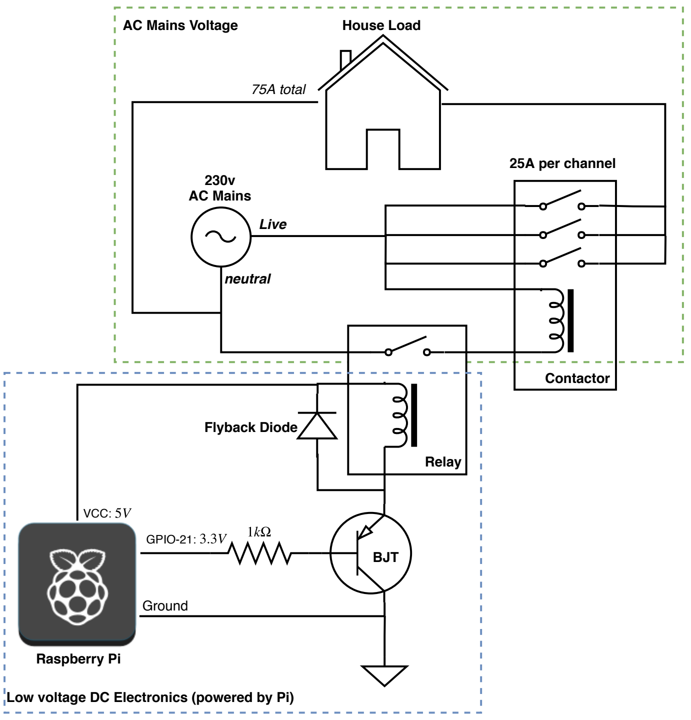


Figure 3 : Circuit diagram showing the implementation of the household load controller

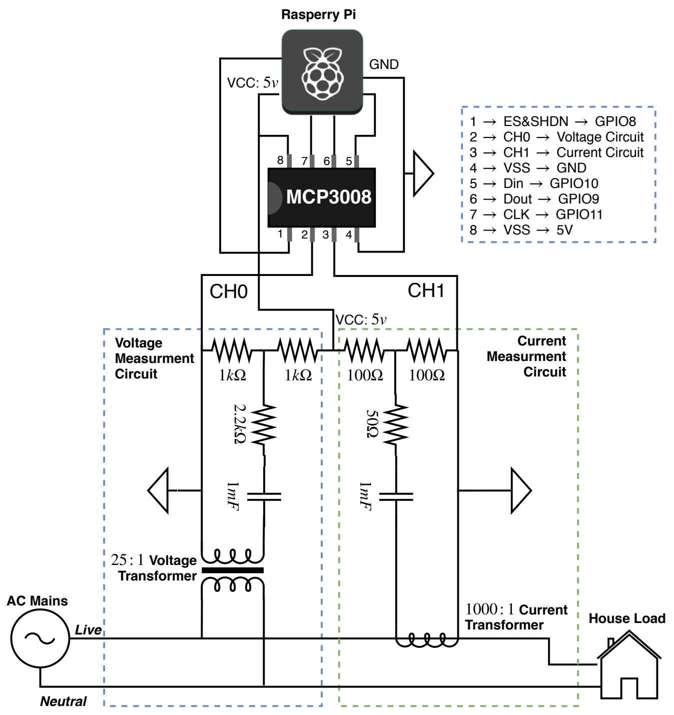


Figure 4 : Circuit diagram showing the implementation of the household load measurement circuitry used to record voltage and current

Note that the relation of in-phase and out of phase corresponding consumption and production of energy respectively are arbitrary. If the voltage or current transformer polarities were flipped around, these definitions would change.

A number of scripts are used to convert the precise ADC readings into the actual voltage and current values experienced by the load. These scripts can be seen here.

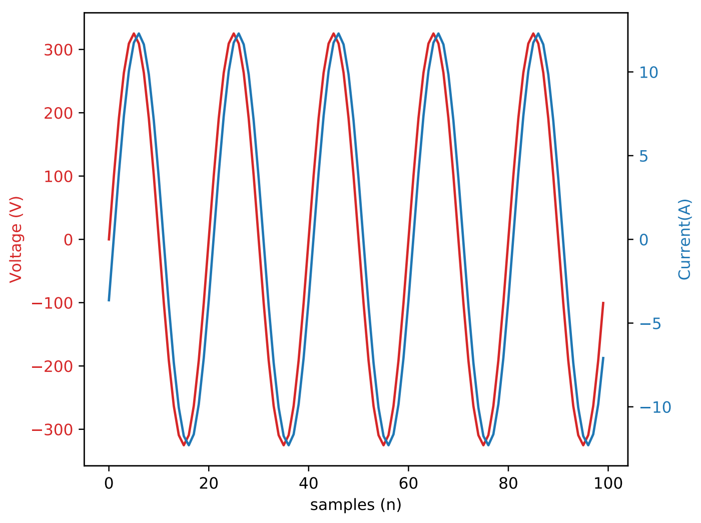


Figure 5 : Measurement recordings produced by the measurement circuit for a 15A household load, showing energy entering the house; in consumption mode

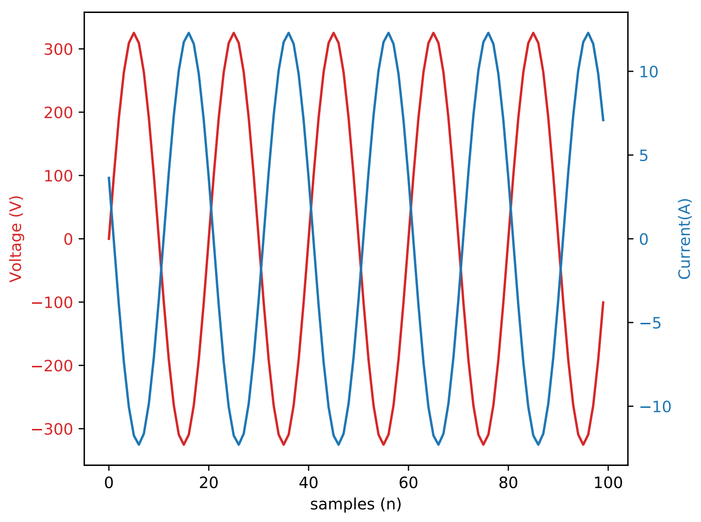


Figure 6 : Measurement recordings produced by the measurement circuit for a 15Amp household load, showing energy leaving the house; in production mode

A5 Python Controlling Scripts

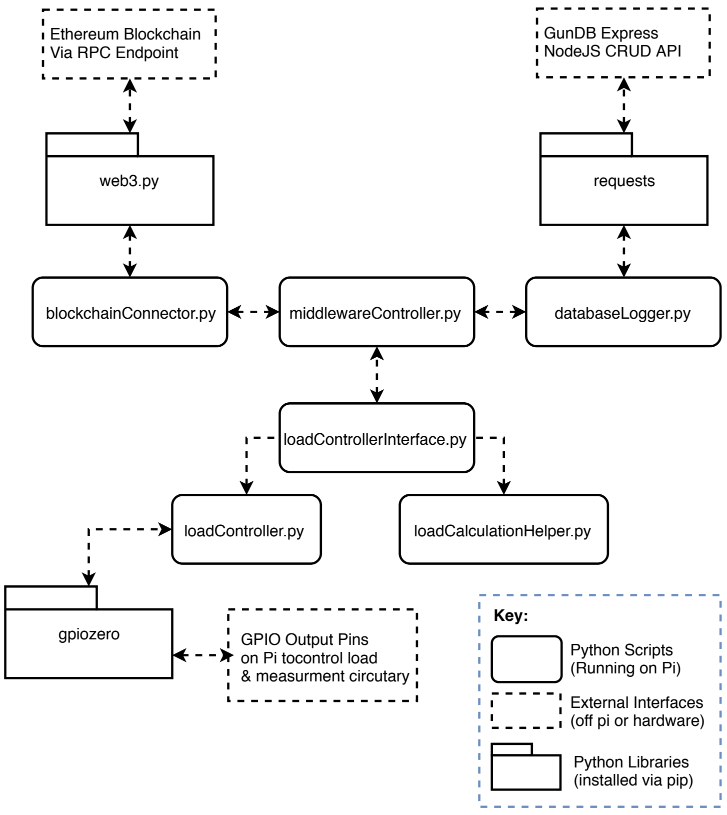
A series of Python scripts were created to control the load state, as well as to interact with the blockchain and distributed database. Figure 7 below indicate these connections. The associated source code can be found here.

The Python code running on the Raspberry Pi's consists of four main sections: `loadControllerInterface.py` is responsible for controlling the house load as well as performing measurements. This script can be seen as the interface to all external hardware.

`blockchainConnetor.py` provides the interface to communicate with the Ethereum blockchain through the use of `web3.py`. This script communicates directly with the smart contracts on the blockchain.

`databaseLogger.py` provides an interface to communicate with the distributed `GunDB` graph database via an `Express/Nodejs` API used to provide live usage metrics of household meters.

`middlewareController.py` provides an interconnection between all three previous scripts and hosts the logic for when and how tokens are minted/burnt. This script calls functions within the other scripts to facilitate all interactions with the both the hardware and the blockchain.



Ethereum Blockchain Via RPC Endpoint
GunDB Express NodeJS CRUD API
web3.py
requests
blockchainConnector.py
middlewareController.py
databaseLogger.py
loadControllerInterface.py
loadController.py
loadCalculationHelper.py
gpiozero
GPIO Output Pins on Pi tocontrol load & measurment circutary
Key:
Python Scripts (Running on Pi)
External Interfaces (off pi or hardware)
Python Libraries (installed via pip)

Figure 7 : Python Controlling Scripts Interconnection Showing how each script communicates with each other

A6 Selection of Token Economics Mechanism

There are a number of different ways to tokenize electricity to interconnect the blockchain with the production and consumption of energy. Each of these methods will be outlined below.

A6.1 Tokenization of the kWh The simplest way to correlate electrical energy to a blockchain asset is to tokenize the kWh using a utility token. This means generating an ERC20 token and relating this to the production/consumption of energy. This is the methodology used by meterBlock's KraG token, wherein each token corresponds to a kWh of energy. This method is outlined in the report above. Additional discussion follows below.

Advantages

- Simple to understand and implement
- Easy to interconnect with existing systems as a standard token can be traded on any normal crypto exchange.
- Has a close link to the actual flow of energy; the process of minting and burning closely reflect energy production and consumption
- Enables people to pre-load a number of tokens onto the meters to provide energy into the future.
- Enables a user to generate power during the day (with solar, for example) then buy energy back from the grid

later on in the day when their production is less than their consumption.
- Usage of tokens does not require any direct involvement by the user; once tokens are loaded, the system continues to operate until all tokens are depleted.
- Price that tokens are traded are defined by supply and demand on normal crypto exchanges meaning that the complex problem of price discovery can be ignored.

Disadvantages

- There is no guarantee that if you have tokens, you can get power; There might not be generation capacity to provide energy for the grid.
  - This is the same as the current system however; just because you have tokens does not mean that your lights will turn on.
- As the energy producers are effectively "mining" tokens on energy creation, this opens the market place up to being manipulated. If a large producer, such as Eskom, held a lot of tokens and chose not to sell them to the open market, the supply would be reduced resulting in an over inflated market price of tokens.
- The price within the crypto exchange does not necessarily correspond to the grid health as these systems are disjoint;
  - The crypto exchange price is defined by market forces(supply and demand)
  - The grid health is defined by the difference between supply and demand of energy
- As a result, the grid can become unsustainable due to too much load on the network and the token price dues not change. To combat this, pricing oracles are suggested wherein the conversion between a token and a kWh changes based on grid load.
- As soon as a pricing oracle is introduced, the notion of a kWh = 1 KraG brakes down; you no longer have a direct conversion between the two and the idea of the "tokenized kWh" no longer applies.
- There is a majour problem with starting the system from black; this is discussed in the report in 8.2.4

Despite the problems with the tokenization of the kWh, it proves to be the simplest implementation that works in both the micro and macro grid configurations. Additionally, as this project aims to be a proof of principle, it can show the validity of the underlying technology without convolving the complexities of economics. While the other implementations do potentially offer more realistic models, they are left for future experimentation.

A6.2 Stable Coins Used to Buy Energy Another implementation involves the creation of a "stable coin" that is then used to buy/sell energy to/from the grid. A stable coin is a crypto asset that corresponds to a real world currency, such as the tokenization of the USD. This is the method that Grid+ uses, through their *Bolt* token[3]. In this implementation, a user would load their meter with stable coins and then buy/sell energy to the open grid at the live market price. Instead of a token corresponding to a kWh and the conversion being a direct 1:1 matching, the current market price of energy will be used in the exchange. This implementation closely mimics the current pre-paid energy meter setup within South Africa where a user could load x number of Rands onto their meter and be billed in accordance with their consumption.

Advantages

- Closely mimics the current South African setup.
- Live billing is simplified as there is a direct conversion between Rands and kWh.

Disadvantages

- Complexity in how people are paid out.
  - If there is a discrepancy between what the producers produce and consumers consume, someone will loose out.
  - One potential solution to this is to use a pooling methodology where at the end of a period of time a "kitty" is created and from this people are paid out proportionally to their contribution.
  - Issue is that this requires a pooling and pay out period to occur at a particular time. If you miss the pooling period, you loose out.
  - Additionally, this will result in higher network traffic due to needing to communicate with the blockchain more frequently.
- Inability to mint/burn a stable coin's means that you cant introduce new funds into the system. As a result, every exchange of energy needs to be a direct peer to peer exchange wherein the buyer of energy have their funds transfered to a seller, directly (hence the notion of the pool)
  - Note that you cant mint or burn a stable coin as each stable coin *must* be backed by a real amount of money in some bank account, somewhere in order for it to be backed. As a result, adding new tokens and liquidity to the system becomes extremely complex.

A6.3 Exchange of Energy Bundles in Smart Contracts
This implementation is the most complex, with the highest number of moving parts. It is also solves many of the issued presented in the other two models.

Simply put, producers sell a set amount of energy to be produced at a set time. A consumer then buys this bundle of energy to be consumed at a set time. Smart contracts are used to ensure that the producer holds up their end of the deal through the use of a collateral mechanism; if the producer said she would produce 100kWh and only produces 75kWh, she looses out on a portion of the collateral. This ensures that there is never more energy sold than what the network can produce as the threat of the loss of collateral will mean that producers won't oversell on their production facilities. This collateral can then be used for one of two things: either it is sent to the consumer who did not get their energy or it is used to buy energy from another producer and then use that energy to fill the end of the deal for the consumer.

Advantages

- Ensure network stability through guaranteed collateral mechanism

Disadvantages

- If a consumer buys x units of energy to be consumed at a set point in time and during this time consumes $x+1$ units of energy, their lights will turn off as they exceeded the energy that they had purchased. As a result, the only logical behavior as a rational actor in the network is to always buy more energy that what you require. At the end of the time period, the consumer then looses this excess energy that they did not use, in a similar fashion to what happens with a phone contract.

- If a producer committed to produce x units of energy to a particular consumer but said consumer only draws $x-1$ units of energy, then the producer will loose their proportion of the collateral due to not producing as much energy as they had committed to consumer.
  - In effect, the collateral mechanism implemented to ensure stability of the network is this solutions biggest downside.
  - If the collateral mechanism is removed, then there is no intensive for the producer to produce the energy they said they would and as such there is no guarantee of any form of energy production.
  - As a result, a rational actor on the network would merely sell as many energy bundles as possible as there is no ramification for not holding up one's end of the deal.
- As a electricity grid is interconnected, there is no way to see what energy from a producer when to each consumer; you can only know what enters or exits a particular house or production facility. As a result, if a producer does not hold up their end of a particular deal, the other producers need to fill in the slack to keep the network stable. Conversely, if a consumer does not consume all the energy they had purchased, then all the producers need to lose a proportion of their collateral as the total net consumption on the grid is less than what was sold for that period of time, across all producers.
- A possible solution to the collateral problem is the use of a rating system wherein producers receive a rating based off their historic ability to producer the amount of energy they said they would.
  - Consumers can then choose to buy energy from highly rated producers.
  - The issue is that now there is no disincentive form merely creating a number of new accounts, selling "bad" energy deals that you never intend on filling and then later creating new accounts when your old ones revive bad ratings.
- The construction of the energy exchange marketplace would be extremely complex as you now have three dimensions to consider when selling an asset: price per kWh, number of kWhs to sell and the time when it would be consumed.
  - A traditional asset exchange, such as those within stock markets and crypto exchange only have two dimensions: price and number. This can be represented as a depth chart showing the number of assets to be sold/bought at a particular price.
  - With the introduction of the third dimension, the notion of an order book becomes extremely complex as one of the dimensions is reduced over time (as it *is* time).
  - This means that orders that are not filled become "stale" and cant be executed on after the fact. This is similar to how some assets can have time limits on when trades can be executed.
- This method also requires active participation in the buying and selling of energy in these bundles.
  - It could be configured such that the meters buy energy at the market price but this still requires active engagement with the platform at a high frequency resulting in much higher network traffic than the other two models.

Ignoring the complexity of the order book and the other apparent issues with this model, the fundamental problem of collateral penalty imbalances make this method unusable.

A7 Smart Contract Design

The smart contract used within meterBlock is based off an inheritance structure that leverages the OpenZepplin standard for the basis of much functionality. Additional functionality has been implemented through the overriding of particular functions, namely that of the mint and burn functions. There are a total of 9 smart contracts used in the design of the meterBlock system. The source code for the smart contracts can be found here. Figure 8 Shows the inheritance structure used.

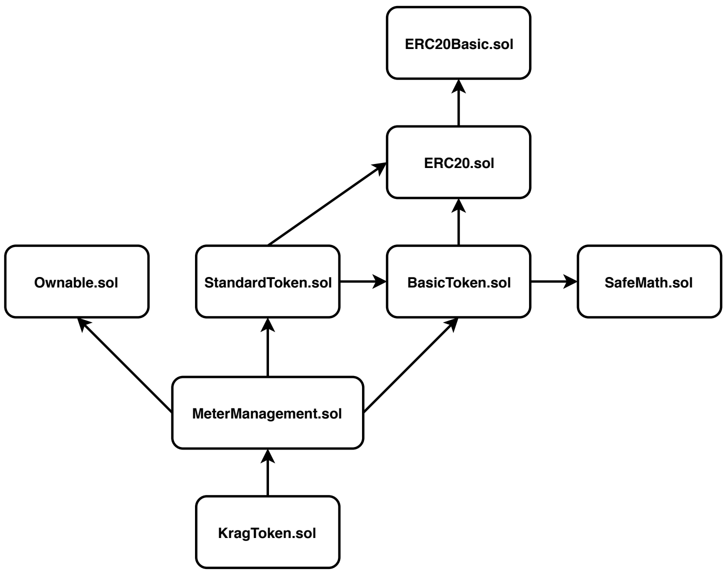


Figure 8 : Solidity Smart contract inheritance structure used in the generation of the KraG token

`KraGToken.sol` is the bottom of the inheritance structure, defining the entry point for all other contracts. This is the base contract upon which all other contracts are deployed. This contract inherits from `MeterManagement.sol`.

`MeterManagement.sol` is the core contract that stores most of the custom logic required to operate the KraG Token. This contract contains all logic to enroll a meter, transfer ownership of a meter and to mint and burn new tokens. This contract inherits from `BasicToken.sol`, `StandardToken.sol` and `Ownable.sol`.

`BasicToken.sol` provides the most basic implementation of a standard ERC20 token, inheriting from `ERC20Basic.sol`. The deployment of `BasicToken.Sol` in conjunction with `ERC20Basic.sol` fully satisfy the ERC20 token standard interface.

`StandardToken.sol` extends the interface of `ERC20.sol` and `BasicToken.sol` to provide additional key functionality like `transferFrom`, enabling a wallet holder to delegate a proportion of their tokens to be accessible from another address. This functionality is utilized on the minting of tokens to give the registered owner of a meter access to the underlying wallet balance, enabling them to withdraw newly minted tokens.

`Ownable.sol` Provides the notion of ownership over a smart contract (or series of smart contracts), providing special permissions and rights for particular user groups. This is leveraged in the registration of new meters, only allowing the meterBlock administrator to have permissions to enroll new meters.

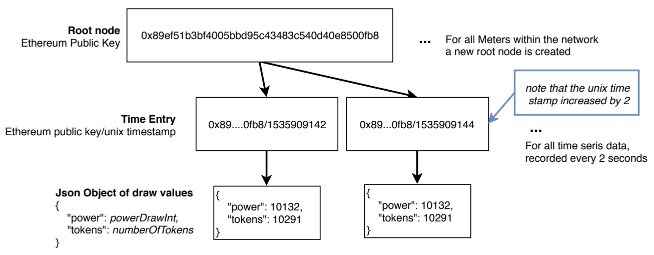
`SafeMath.sol` is used to ensure that all mathematical operations preformed are "safe", in accordance with the OpenZepplin standards. For example, upon any addition or subtraction of tokens, the contract first checks that no overflows will occur, resulting in erroneous contract behavior.

A8 GunDb Graph Design

GunDb is a graph based database, meaning that it uses a graph based structure for semantic queries within a nodal structure. Edges within the graph represent and store data. This differs from a traditional SQL database wherein rows correspond to entries within the database or where Json blobs represent data within noSQL databases.

The graph structure used within meterBlock's decentralised graph database starts with the root node corresponding the the meter's public key. Each meter has its own unique public key as each meter has it's own Ethereum address. From this each entry, generated every two sections, is a child node from this root in the format of `key/unix_timestamp`. Below each of these `key/timestamp` values is the recorded values of: power draw at that point in time and the number of tokens within the users wallet. This structure can be seen in figure 12.

When the user requests information from their particular meter, the client browser synchronizes this particular dataset based off the root node. This means that only the relevant information is sent to the client at any point in time.



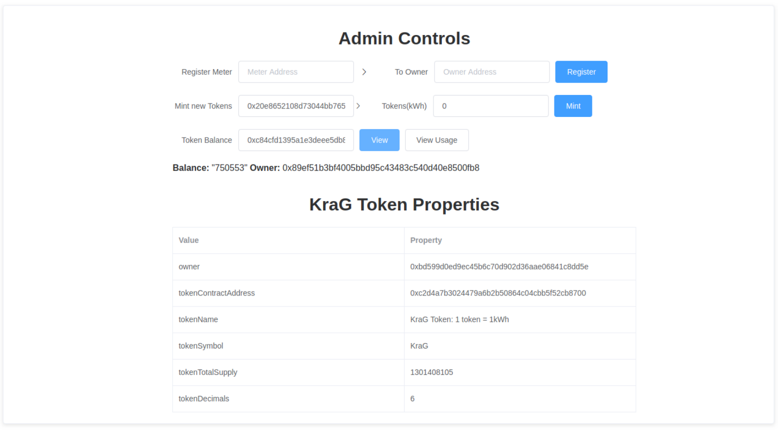
Root node
Ethereum Public Key
0x89ef51b3bf4005bbd95c43483c540d40e8500fb8
...
For all Meters within the network a new root node is created
Time Entry
Ethereum public key/unix timestamp
0x89....0fb8/1535909142
0x89...0fb8/1535909144
note that the unix time stamp increased by 2
...
For all time seris data, recorded every 2 seconds
Json Object of draw values
{ "power": powerDrawInt, "tokens": numberOfTokens }
{ "power": 10132, "tokens": 10291 }
{ "power": 10132, "tokens": 10291 }

Figure 9 : Graph database structure used in the decentralized database to record live usage metrics of the meters. Note that this structure repeated for every node in the network

GunDB does not store any value and as such, it's security requirements are much lower than that of the blockchain. The two technologies can be seen to complement each other; Ethereum for it's security and immutability, used as a store of value and GunDB for its speed, scalability and accessibility, used for real time information.

A9 Application frontend

A user facing frontend was created to enable the user to interact with the smart meters. The frontend code can be found here. A number of screenshots were taken to show what the User interface of the final product looked like. These can be seen below.


Admin Controls

Register Meter: Meter Address > To Owner: Owner Address [Register]

Mint new Tokens: 0x20e8652108d73044bb765 > Tokens(kWh): 0 [Mint]

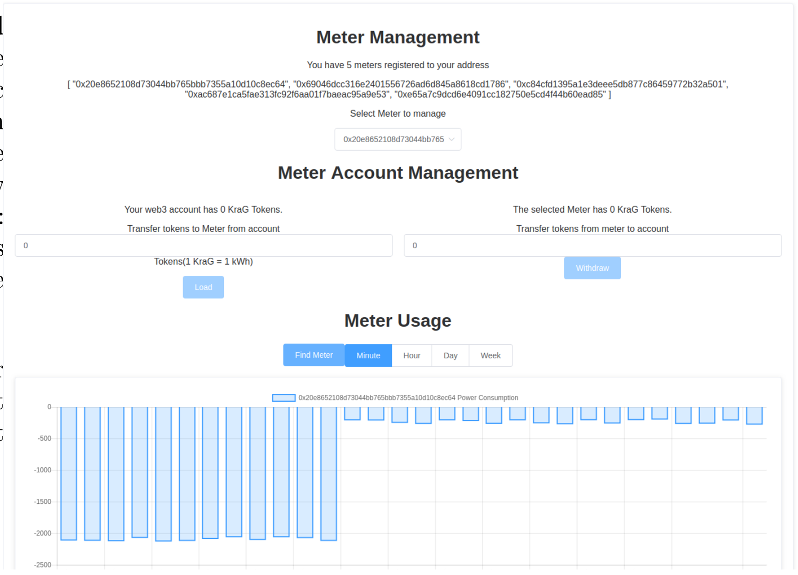
Token Balance: 0xc84cfd1395a1e3deee5db8 [View] [View Usage]

Balance: "750553" **Owner:** 0x89ef51b3bf4005bbd95c43483c540d40e8500fb8

KraG Token Properties

Value	Property
owner	0xbd599d0ed9ec45b6c70d902d36aae06841c8dd5e
tokenContractAddress	0xc2d4a7b3024479a6b2b50864c04cbb5f52cb8700
tokenName	KraG Token: 1 token = 1kWh
tokenSymbol	KraG
tokenTotalSupply	1301408105
tokenDecimals	6

Figure 10 : Application administration frontend used to enroll new meters as well as other admin functions


Meter Management

You have 5 meters registered to your address

["0x20e8652108d73044bb765bbb7355a10d10c8ec64", "0x69046dcc316e2401556726ad6d845a8618cd1786", "0xc84cfd1395a1e3deee5db877c86459772b32a501", "0xac687e1ca5fae313fc92f6aa01f7baeac95a9e53", "0xe65a7c9dcd6e4091cc182750e5cd4f44b60ead85"]

Select Meter to manage: 0x20e8652108d73044bb765

Meter Account Management

Your web3 account has 0 KraG Tokens.

Transfer tokens to Meter from account: 0

Tokens(1 KraG = 1 kWh)

[Load]

The selected Meter has 0 KraG Tokens.

Transfer tokens from meter to account: 0

[Withdraw]

Meter Usage

Find Meter
Minute
Hour
Day
Week
0x20e8652108d73044bb765bbb7355a10d10c8ec64 Power Consumption
0
-500
-1000
-1500
-2000
-2500

Figure 11 : Application meter management frontend that a user would used to interact with one of their meters. These values are negative, indicating the production of energy. The steep drop in recording values corresponds to the change of production state

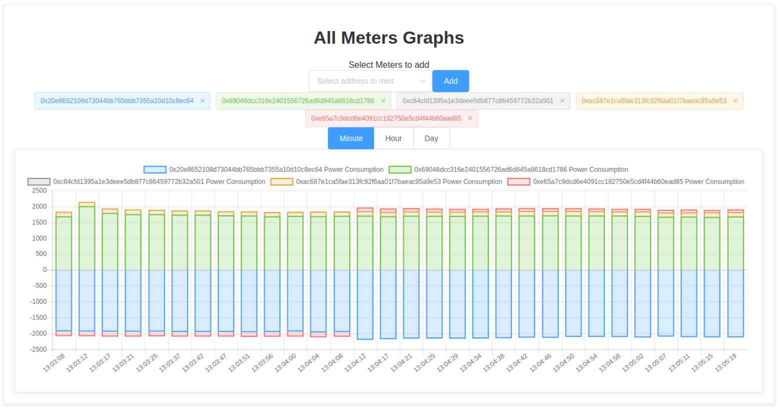


Figure 12 : Microgrid network health showing a number of meters interacting with the system. Positive values correspond to the consumption of energy and negative values correspond to the prosecution of energy. The graph shows half way through the time series the hybrid load switching from producer to consumer.

A9.1 Frontend Technology Stack The frontend was created using vuejs and elementUI to provided a reactive, responsive, modern user interface. Web3 connectivity was provided to interact with the blockchain through web3.js. A user can then connect to the blockchain using Metamask or a web3 enabled browser, such as Status. Chars.js was used to provide graphical visuals of network utilization. Lastly, vue-gun was used to provide the interconnection between `vuejs` and `gundb`.

A10 Microgrid Test Setup

A micogrid of five smart meters was constructed. Diagram 13 represents the testing setup used in the construction of the micro grid. Photo 14 and 15 show images of the actual testing implementation.

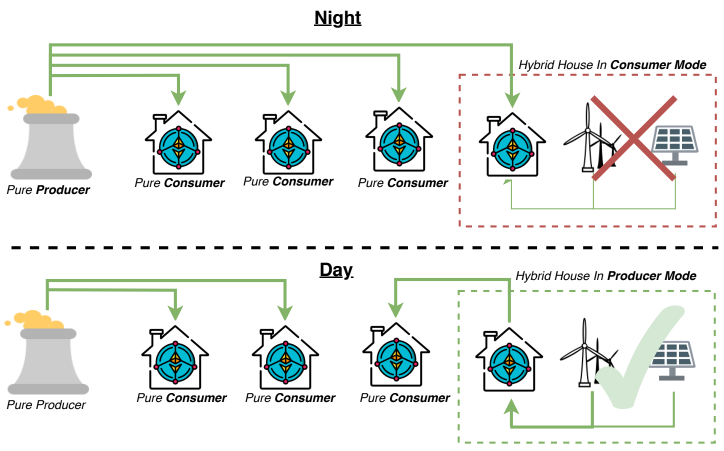


Figure 13 : Microgrid testing setup used to benchmark meters under a real world setup

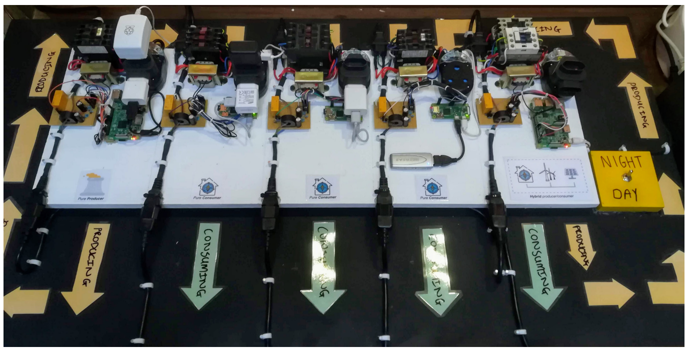


Figure 14 : Photograph showing the testing setup of the microgrid

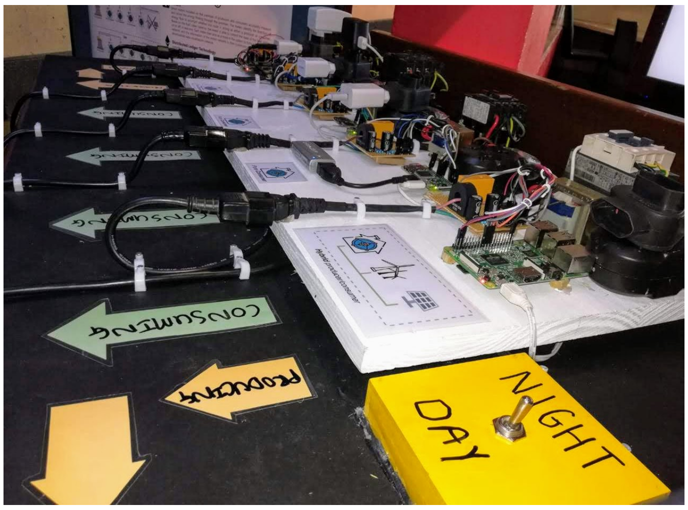


Figure 15 : Photograph showing a close up image of the microgrid

B Reflection on Group Work

All in all, I enjoyed working in the group I was part of. Brandon and I are close friends and have done a number of projects together before and the synergy that we've experience before continued into this project.

meterBlock proved to involve a lot of hardware in the construction of the meters as well as software to control the meters and interact with the blockchain so having someone dedicated to working on the hardware while I worked on the software ended up working really well. In the beginning of the project, we did most of the work together; we designed the original smart contract together as well as all iterations of the physical meters. Additionally, all design decisions around the tokenization of energy and other subtleties were team decisions.

As the project progressed and the idea became more formalized, a clear division of labour formed. For the last three weeks of the project, I worked exclusively on software and Brandon did exclusive hardware. This worked really well as we were both able to play to our strengths and we were able to make something that we were both proud of.

One source of contention was the location of where work should be done. As much as possible, we worked on campus. However, the wits wifi is notoriously bad and proved to be unusable for a large majority of the project. As a result, a lot of time was spent at my house where we continued to work on the project, often into the evenings. This posed a small problem as I had to often fetch Brandon from campus and bring him to my house in the morning and take him back to campus in the evenings. This turned out not to be that much of an an issue at the end and Brandon spent a number of evenings working at my house during the course of the project.

I myself am an Information Engineer, while brandon is an Electrical Engineer. Our project was perfectly suited for the hybrid between information and electrical engineering, with the merger between and software and hardware so apparent. I think we both greatly benefited from having each other in the group.

There was minimal conflict during the course of the project with both Brandon and I valuing and considering each other opinions. I look forward to being able to work on future projects with Brandon.

School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
ELEN4002/4012: Project Specification Outline

To be completed by supervisor
Assessment:
☐ Unacceptable ☐ Poor
☐ Acceptable ☐ Good ☐ Excellent

Project Title:	Blockchain based tokenised decentralised grid network (meterBlock)		
Group Number:	18G35	Supervisor Names:	Viv Crone and Mitch Cox
Member 1 Name:	Brandon Verkerk	Student number 1:	875393
Member 2 Name:	Christopher Maree	Student number 2:	1101946

Project Specification:

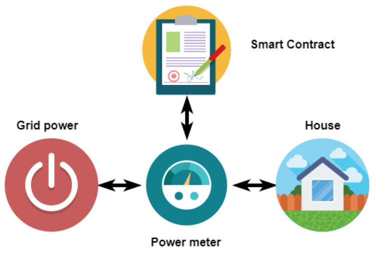
meterBlock provides a decentralised platform to facilitate the governance of electrical energy consumption and production, measured and controlled by pre-paid IoT power meters. meterBlock consists of three distinct pillars:

1. Utility tokens (a blockchain store of value, representing a real world asset), each representing a kWh, are used to account for the energy produced and consumed on the grid. These "KraG" tokens are the economic mechanism used to control energy through IoT based pre-paid power meters. The meters will measure the energy consumed/produced, and control its supply to the load/grid.
   - An IoT device will be used to control a relay or contactor to facilitate and interrupt power supply to a load.
   - An Ethereum Smart contract will be used to manage the utilisation of KraG tokens according to production and consumption of energy
     - This token is based on the ERC20 token standard and is allocated on the production of energy and is deallocated on consumption of energy
     - A website dashboard will be created to facilitate user interaction with the smart contract
   - The IoT device must interact with the smart contract to:
     - Enable the deallocation of tokens, representing the consumption of energy. When these tokens run out, the power turns off
     - Enable the allocation of tokens, representing the production of energy. These tokens are then transferable.
   - *The deliverables related to this pillar are in milestone 1 to 5, with extended implementation up to milestone 7*
2. Using game theory, non-linear pricing curves and other economic incentives will be used to encourage mutually beneficial utilisation of the grid between produces and consumers.
   - By monitoring the load on the grid, the cost per kWh can be adjusted dynamically to encourage advantageous behaviour.
   - *The deliverable related to this pillar is outlined in milestone 8*

Point 1 and 2 above will be implemented without considering how tokens are converted to fiat. The limitation is that no new members can join this system if they don't produce energy. Likewise, members that produce more than they consume are not compensated accordingly.

3. A fiat-to-token exchange is created to facilitate the buying and selling of KraG tokens between consumers and producers.
   - *The deliverable related to this pillar is outlined in milestone 9*
4. Microgrid logic is included to facilitate dynamic load balancing within the grid as well as cost savings for the end user.
   - Each household appliance will be given a priority, enabling intelligent load shedding (turning off of a geyser, before your lights).
   - Through the economic platform from (3), the smart meter can vary its cost model based on user specifications, by turning off high draw, low priority loads when energy is expensive.
   - *The deliverable related to this pillar are outlined in milestone 10*

Points 1 and 2 above is defined as the minimum viable deliverable for the project and are considered the base success criteria. Implementation of point 3 and 4 are seen as stretched goals. Point 3 has inherent challenges associated with it such as normal market forces and human sentiment. Implementation of point 3 requires assumptions to be made that will be addressed in the report. Basic overview of proposed system:



Milestones:

1. Ethereum smart contract capable of creation, allocation and deallocation of KraG tokens (self-standing entity, not linked to anything at this point)
2. IoT microcontroller (Rasberry Pi or Arduino) capable of measuring power consumption (meter).
3. Meter capable of controlling (switching on and off) the load.
4. Interaction between meter and smart contract (connection of meter to the blockchain)
    - This interaction must enable point 1 and 2 (consumption of power results in reduction of tokens. When tokens run out, power turns off).
5. Steps 1 to 4 are then implemented for the production of power
6. User interface implemented on the meter to:
    - Display the real time current token balance of the meter
    - Display of key power metrics(total energy produced/consumed, contract information, tokens consumed, etc.)
    - QR code displayed directly meter enabling tokens to be sent to the meter from a mobile device (loading more tokens)
7. Creation of a web-dashboard to view the current status of meters on the blockchain and manage the smart contracts

These milestones are for specification part 1 above, where the economic forces, such as the price per token and how they are traded, will not be initially taken into account.

8. Non-linear pricing curves implemented through the use of a price oracle.
    - This means that the price of energy changes according the load on the grid

At this point, a fully implemented MVP will have been been created, that could conceivably work in the real world.

9. Simple token market, enabling the buying and selling of tokens for fiat (or other crypto assets)
10. User able to specify individual appliance load priority within her household
    - Smart load sheading enable to selectively turn off low priority devices under high grid load
    - Optimised cost mode to schedule low priority devices to only run when the price of power is low

Budget/resources:

1. Arduino or Raspberry Pi to act as the power meter (R400)
2. Relay or contactor to enable load control (R200)
3. 4 Clip on Current transformers to measure power produced/consumed (R200)
4. LCD user interface for meter (R300)
5. Breadboard ,pcb, wires and other miscellaneous electronics (R100)

Total: R1200

Risks / Mitigation:

Scope creep is a major problem with our idea. As such, we have been careful to define the minimum deliverables (pillar 1 and 2 above), and will only work on the future extensions of the project once these are completed and documented in their entirety. Each component within the project is complex; be it the IoT power meter, the smart contract or token exchange. As a result, design decisions have been kept as simple as possible with the intension of keeping the project as a proof of concept.

Risk	**Mitigation**
Implementation of ERC20 token contract proves too complex for implementation	Use a modified version of existing token standard
Inability to implement power meter user interface via LCD screen	Communicate with PC over USB
Inability to link meter to smart contract via API	Simulate connection via MATLAB
Complexity in implemented non-linear pricing curve oracle to inform meters of current prices	Hard code pricing curves into meters, based on voltage
Fiat to token exchange proves too time complex to implement	Simulate an exchange in MATLAB
Economic game theory relating to the transfer of tokens between a consumer and producer outweighs the intended use case value of the project	Treat economics as a black box, making fundamental assumptions to simplify the model
Inter-meter dynamic load balancing microgrid logic communication mechanism proves complex	Simulate a communication network between multiple meters in MATLAB

meterBlock

Supervisors: Mr. Viv Crone
Mr. Mitch Cox

Blockchain facilitated power exchange for microgrid applications

Brandon Verkerk – 875393 Christopher Maree-1101946
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2050, South Africa

16-07-2018

Executive summary:

meterBlock is a collaborative project that aims to provide a decentralised platform to facilitate the governance of electrical energy consumption and production, measured and controlled by Distributed Ledger Technology (DLT) connected power meters. The goal of the project is to build a Proof-of-Concept (PoC) prototype device running on the Ethereum blockchain to facilitate this energy governance. The project commenced on the 1st of July 2018 and will run to completion eight weeks later on the 26th of August 2018.

A primary use of this technology is to provide a platform of energy exchange between producers and consumers within self-sustaining micro-grids, trading surplus energy between one another. Full scale implementation involves replacing the current national grid's management system, whereby each individual entity can act as both a consumer and producer, governed by the blockchain, enabling a novel, trustless, decentralised, energy exchange marketplace. These entities could include, but are not limited to: pure producer - such as power utilities (Eskom) - pure consumer - such as manufacturing plants or traditional households - or producer/consumer - such as gated communities, households, hospitals, shopping centres etc. with installed sources of renewable energy.

The project will run on an Ethereum based blockchain (such as Quorum) and will implement the tokenization of an energy unit through an adaptation of the ERC20 token standard.

meterBlock lays the foundations for future collaborative work when taken to its logical extreme at a national grid level. This report discusses the inherent issues that must be considered before full scale implementation of a DLT-based system can be put into production. Some of these issues relate to the practicalities and complexities of implementation when considering application at a national grid level, but can also be extended to the legal and regulatory factors and their broader economic impacts.

Key Words

Below are a collection of application specific key words and jargon used throughout the report.

A. Application Program Interface (API)

A collection of functions and procedures that define an interface to access and control communication between various components.

B. Blockchain

A Blockchain is a public, peer-to-peer, digital ledger, used to record transactions chronologically. It represents an immutable, persistent record that can never be altered or changed by anyone due to the distributed nature of the consensus algorithm. There is no central authority figure due to the decentralised nature making it highly censorship resistant.

C. Blockchain Node

A node is a computer connected to the blockchain that stores a full copy of the historic ledger.

D. Consensus

The agreed state of the ledger based on the cumulative communication between the nodes within the blockchain based on a particular protocol, such as Proof-Of-Work.

E. Decentralized

System without central governance. There is no centralized authority that makes decision on behalf of all other parties.

F. Distributed Ledger Technology

A distributed ledger is a database that is shared and synchronized across a network spread throughout multiple sites with an underlying governance mechanism to ensure continued consistency.

G. ERC20 Token Standard

ERC20 is one of the technical standards used for the creation of tokens governed by smart contracts on the Ethereum blockchain.

H. Ethereum

Ethereum aims to create the ultimate protocol for building decentralised applications. Ethereum implements a Turing-complete programing language built into its blockchain, enabling anyone to create and deploy smart contracts

I. Hypertext Transfer Protocol (HTTP)

HTTP is a Communication and transfer protocol based off the client server model to facilitate direct communication between two entities used within the internet.

J. Immutable

Persistent storage of information that is unchanging over time. Records within the blockchain are considered Immutable as it is practically impossible to change them once they are recorded.

K. Ledger

An accounting mechanism used to keep a chronological list of transactions and interactions.

L. Message Queuing Telemetry Transport (MQTT)

MQTT is a data centric communication protocol that uses a client server publish/subscribe pattern. It is designed to be as lightweight, fast and reliable as possible.

M. Metamask

Metamask provides the ability to connect a standard web browser, such as Google Chrome, to the Ethereum blockchain via a predefined RPC endpoint.

N. Middleware

Application to connect two different communication protocols over a network.

O. Off-chain

Information that is not stored on the blockchain and as such cannot be controlled or accessed by blockchain entities, such as smart contracts.

P. Oracle

A mutually agreed upon mechanism used to provide information from the real world (off-chain) to an on-chain asset such as a smart contract. An example could be the oil price.

Q. Censorship Resistance

The ability of the system to prevent third party modification to the ledger.

R. Smart Contract

Self-contained entity that resides on a blockchain that directly controls the transfer of digital currencies or assets between parties under predefined conditions. They can be used to perform computation in a decentralised fashion, without any central controlling entity.

Abstract— **This paper serves to document the development of meterBlock - a decentralised platform to facilitate the governance of electrical energy consumption and production, measured and controlled by Distributed Ledger Technology (DLT) connected power meters - from conceptualisation through to implementation. As a prototype, meterBlock makes use of a Sonoff POW to monitor and record the energy consumption or production of a typical household, and communicates this information via MQTT to a Raspberry Pi Ethereum node. Each section within this report corresponds to the completion of a weekly milestone over the project duration, starting from the 1st of July 2018 until intended completion on the 26th of August 2018. Included too is the preliminary research and key design decisions - such as the choice of MQTT and the Ethereum blockchain - taken before development began, as well as a critical evaluation of the intended system as a whole with regards to implementation and price discovery.**

I. Introduction

Since the first enquiry from the national power utility (Eskom) in 1989 for the introduction of prepaid power meters, prepaid electricity has become a popular and convenient means of power management in many South African households[1]. meterBlock introduces a novel, scalable alternative energy management system, capable of addressing the consumption and production of energy at a national grid level through the use of DLT. This report covers contextual justification for the implementation of meterBlock, as well as the ideological advantages of the underlying protocol. It then discusses a detailed implementation plan that can be followed to achieve the final product.

II. Implementation Context

In order to fully justify meterBlock's intended use case, a critical analysis of the current system's operations and limitations is required.

A. South African Power Market

There are currently only two ways in which to purchase electricity in South Africa.

1) Pay-as-you-go model: consumers of electricity receive a bill at the end of the month corresponding to their energy usage for that month. This requires a representative of the national utility to visit the household to conduct a reading on the household's power meter. However, these readings are taken on an infrequent basis and as such are inaccurate from month to month as an averaging process is used. This means that on one month you may pay for more than you actually consumed and on another the converse may be true.
2) Pre-paid model: addresses the inaccuracies inherent to the pay-as-you-go model as funds are preloaded on the meter and the user is only billed as-and-when a kWh is consumed.

However, neither of these models facilitate the ability to sell power back to the national grid. This does not provide additional incentives for the installation of renewable energy systems within the household (or other consumers) as there are no mechanisms to profit from surplased energy produced. Additionally, the user has to trust the accounting mechanisms used by the national utility. This includes the manner through which energy consumption is recorded and the associated billing thereof.

This inherently means that there is no platform upon which a microgrid can be created, wherein energy exchange occurs between producers and consumers. Moreover, the current systems have a central point of failure due to being limited to having only one energy producer.

B. How meterBlock Addresses the South African Power Market's Limitations

As identified before, there are two fundamental problems with the current South African power market that meterBlock aims to address: naimly the ability to sell power back to the grid and the requirement to trust a central authority in the accounting and billing of energy. Both of these problems can be addressed through the implementation of a DLT governed power metering system.

The ability to sell power back to the grid is achieved through the use of autonomous smart contracts that govern and control the flow of tokens in accordance with energy consumed and produced. These smart contracts reside on a distributed blockchain and as such have no central controlling authority making the whole system inherently trustless. This results in the ideological notion of trustless governance. Additionally, this governance mechanism provides the ability for anonymous, transparent auditability of the billing ledger.

This fully justifies the exploration into the design of a DLT energy governance system. What follow is a detailed project plan that can be used to fully realise the minimum viable prototype.

III. High Level Proposed System Overview

The proposed system consists of 5 discrete components:

1) Toggle switch and energy measuring device: A device is required to control supply to a house (in consumption mode) or conversely control supply from a house to the grid (in production mode). Additionally, monitor and record energy consumed/produced by the house. In the proposed prototype, this is achieved with a Sonoff POW.

2) Communication protocol to connect toggle switch to Ethereum node: A communication protocol is required to convey all information from the toggle switch energy measuring device (Sonoff POW) to the blockchain as well as commands from the blockchain to the toggle switch. In the proposed prototype, this is achieved through use of a MQTT broker running on a Raspberry Pi Zero W.

3) Ethereum node and middleware: An access point is required to transmit the received information from the communication protocol to the blockchain. For this, a full ethereum node will run on a Raspberry Pi Zero W. Additionally, middleware is required to translate messages received view MQTT into information transmitted to the blockchain. In the proposed prototype, this will also run on the Raspberry Pi Zero W and will be written in Python using `Web3.py`.

4) Smart Contract Management System: A smart contract governance system is required to manage the allocation and deallocation of tokens based on consumption and production for each consumer/producer. In this implementation, this information is received via the middleware running on the Raspberry Pi Zero W.

5) User front end: A simple user front end dashboard is required to inform the user of their consumption/production as well as provide a mechanism to load/withdraw tokens. In the proposed prototype, this is achieved with a `vue.js` front end using `web3.js` to interact with the blockchain.

An overview of this proposed system can be seen in the Figure 1. This diagram shows the communication between households, Sonoff POW's and a node.

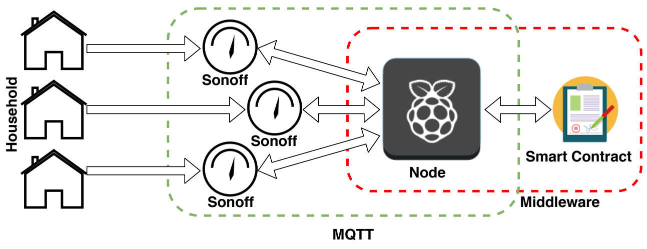


Fig. 1: MQTT System Diagram

This deliverable does not provide the complex economic mechanism to facilitate an energy exchange marketplace but rather provides the underlying governance protocol upon which this mechanism can be built. This economic mechanism is left as a discussion point for future consideration. The final intended prototype deliverable can, in theory, be used to replace the current pre-paid power model without alteration, providing a token based governance mechanism similar to that already used by the national energy market.

IV. Project Plan

The project plan is broken up into eight week-long sub-sections. A holistic overview of the project plan can be viewed in conjunction with the work breakdown structure and associated gantt chart outlined in appendix 2.A and 2.B.

A. Preliminary Research (01-07-2018 to 15-07-2018)

Before any implementation work can begin, a holistic overview of the surrounding technologies is required. This section outlines areas of interest to conduct research regarding: prepaid energy meters, blockchain projects of interest, IoT devices and the blockchain as well as real world applications of this technology.

1) Existing Blockchain related Energy Metering: A number of projects have been implemented using blockchains as auditing and accounting mechanisms for energy consumption and production. One notable project is Grid+, a project run by Consensys[2]. More information on Grid+ can be seen in appendix 1.A.

While Grid+ does have some similarities as the intended use case of meterBlock, they do not offer the ability to sell power back to the grid. Additionally, they do not offer any notion of energy exchange and are simply recognised as another power reseller. More over, their implementation would, strictly speaking, not work in the South African context as Grid+ requires the ability to sell power to consumers over the national grid. At present, one is not able to do this over the South African power grid as South Africa has one national power utility that does not accommodate for individual households to sell back to the National Power Utility (Eskom) [3]. As noted, this is too is a problem for meterBlock when the intended use-case is taken to its logical extreme, and as such the system should not be designed for this scenario alone.

2) Remote Switching Technology: A remote controlled switch, containing a relay or contactor, will be used to simulate the house power control mechanism. Future implementations of meterBlock could involve the design and construction of a high current controller, able to safely control the supply to a full household load.

For the purposes of this project, a Sonoff POW has been selected as it enables remote switching of a load along with remote energy consumption monitoring of said load[4]. The Sonoff POW connects to standard WiFi networks and can be reprogrammed to accommodate custom firmware and functionality. The relays can accommodate up to 15A and the device runs off of 220v mains supply without the need for an external power supply. In addition, the Sonoff POW is compliant with ICASA and the council Radio Equipment Directive (RED) 2014/53/EU making it acceptable to use in a household context[5].

3) Power Consumption Metering Technology: Current Clamps could be used to monitor the power consumed by the load, but for the sake of simplicity the internal power meter within the Sonoff POW will be used to monitor the load draw.

4) Selection of Appropriate Firmware: The stock firmware that comes with the Sonoff POW enables remote control of the application over Hypertext Transfer Protocol (HTTP) and Message Queuing Telemetry Transport (MQTT)[6], [7]. However, this firmware does not offer direct access to the relays over an API and requires the use of a verified application, such as eWeLink[3]. As a result, a custom firmware should be installed on the Sonoff to accommodate additional functionality. For the purposes of this project, Tasmota has been selected as the appropriate firmware. For an extensive justification of this decision, see appendix 1.B.

5) Selection of Appropriate Communication Protocol: A communication protocol is required to interact directly with the Sonoff's. This protocol needs to be lightweight, maintainable and scalable. Two such protocols are HTTP and MQTT, both supported by the selected custom firmware, Tasmota. MQTT offers a clear advantage over HTTP for the application of this project and as such has been selected as the communication protocol of choice. A basic diagram of the publisher subscribe model is shown in the figure 2 below. For the full justification for this choice, see appendix 1.C.

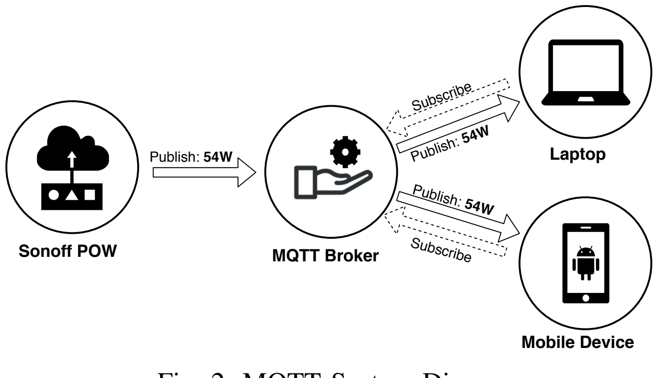


Fig. 2: MQTT System Diagram

6) Blockchain Related Research: A blockchain is required to facilitate decentralised, trustless energy consumption bookkeeping and auditing. A number of public and private blockchains are acceptable for this use case. For the purpose of this project, a private testnet will be used to simulate a consortium between trusted parties to create a permissioned blockchain. For this project, Ethereum will be used as the blockchain of choice. For a full justification, see appendix 1.D.

7) Connection of IoT Device to Blockchain: Next, one needs a way to connect the Sonoff to the blockchain. In an ideal case, with maximum decentralization attaining the highest degree of censorship resistance, each house within the microgrid is provided with a full Ethereum node. These nodes would attest their associated meter's energy consumption to the blockchain. In this implementation, the node itself implements some form of middleware discussed later on in this report, to bridge the off-chain and on-chain data sources of the Sonoff POW and blockchain logics.

8) Monitoring IoT Device From Blockchain: An Ethereum smart contract needs to have access to off-chain information in order to perform correct power consumption and production bookkeeping. Traditionally, this poses a problem as a blockchain is inherently trustless (by design) but an off-chain data source comes from a central entity and as such is not trustless. The normal blockchain approach is to use a centralized Oracle that is agreed by all vested parties to be trustworthy to provide information to the blockchain. The implementation required by meterBlock is, however, slightly different. The individual nodes themselves are the only entities that know about the power consumption of the devices that they connect to. There is no other source of truth, other than the nodes themselves.

This poses a problem such that nodes can be tampered with and as a result can report erroneous power consumption/production readings to the blockchain. There is no way for the other nodes to verify the integrity of the reported readings by the other nodes.

At this point, a fundamental assumption needs to be made: no one can ever modify or change the power meters. They are immutable and always report the correct power readings to the blockchain. This assumption is already made in existing prepaid power meters as there is a risk that individuals can bypass the meter or tamper with readings.

This assumption must also hold when considering the middleware as this can also be seen as a tamperable component of the communication protocol. For the purposes of this project, this will be in the form of a Python application using web3.py to interact directly with the blockchain. This python application will run on each management node and will be responsible for reading information directly from the meters(via MQTT) and transmitting transactions to the blockchain to interact with the management smart contracts.

9) Controlling IoT Device Using Smart Contract Logic: Controlling an off-chain device using an on-chain smart contract involves the nodes monitoring the state of a smart contract and then controlling the Sonoff POW in accordance with the output of the smart contracts. This will occur through the same middleware previously proposed wherein web3.py will be used in conjunction with an MQTT interface to control an off-chain asset with on-chain logic.

B. Preliminary Hardware Experimentation (16-07-2018 to 21-07-2018)

Once the appropriate hardware has been researched and selected, an understanding of the mechanisms through which the remote switch operates must be understood to fully capitalize on its functionality for the intended use case. This includes the purchasing of the intended remote switch and accompanying logic controllers, and experimenting with the stock firmware and user interface of the remote switch. Thereafter, the chosen custom firmware can be flashed onto the device and experimented with to ensure correct operation of the device before implementing the desired communication protocol.

C. Purchasing Selected Hardware

As per section IV.2, the selected remote switch and power measurement device is the Sonoff POW - a wifi enabled 15A relay switch that comes pre-configured with an energy consumption reporting mobile application. Three Sonoff POWs should be purchased to correctly simulate the microgrid application. In addition to the Sonoff POW, a Raspberry Pi Zero W single board computer (SBC) must be purchased to host the MQTT server that will facilitate communication and control between the Sonoff POW and the chosen blockchain. In the event that a custom device will be built to use instead of the Sonoff POW, additional hardware that includes current clamps and contactors must be purchased, however, these will not be used during the proof-of-concept phase of the project and can be neglected for the time being.

Preliminary research suggests that Micro-Robotics is the most cost effective distributer of all desired hardware components[8].

1) Experimentation with remote switch: To understand the operation of the Sonoff POW, a light bulb is connected to the device so that it can be toggled remotely through its preconfigured mobile application. Once connected, the device can be configured to the desired WIFI network by following the instructions outlined in the enclosed leaflet that comes with the device, and downloading their proprietary mobile application eWeLink[5].

Once configured, the state of the relay should be controllable through eWeLink, and the current power consumption, voltage and current draw should be visible through the app. Note, the Sonoff POW registers no energy consumption when connected to a energy saving CFL lamp due to its limited current draw. As such, a standard 50W incandescent lamp should be used to properly monitor and record energy consumption.

2) Flashing of Selected Firmware Onto Remote Switch: The stock firmware of the Sonoff POW and accompanying mobile application does not have a built in web API that can be used to relay the state of switch and power related data back to the blockchain. As a result, custom firmware must be flashed onto the device to provide the functionality necessary for use in the desired application. As per section IV.4, Tasmota is an open source alternative firmware variant for ESP8266 based devices such as the Sonoff POW. To leverage the functionality of Tasmota and the HTTP endpoint or custom MQTT broker, the device must be flashed with the custom firmware. To do this, a standard FTDI flashing module breakout board must be purchased (also available from Micro Robotics), and connected to the Sonoff POW. After downloading or cloning the git repo, the Sonoff POW can be put into flashing mode by connecting inserting the flashing module into the computer with the Sonoff POW toggle button held down. Consulting Tasmota's GitHub page presents the complete set of flashing instructions, including how to edit the config file to connect the device to the chosen network after flashing[9].

3) Experimentation with Custom Firmware: Once flashed, the device can be reconnected to the light and the custom web interface provided by Tasmota can be used to toggle the relay, as well as monitor the power consumption of the device connected to the Sonoff POW.

4) Access through HTTP API endpoint: Through the use of the API endpoint provided via Tasmota, one should be able to exert complete control over the Sonoff pow. Predefined endpoints are outlined in the Tasmota documentation and can be tested for full functionality.
This functionality includes the ability to:

- Control the relay state (toggle or set to a predefined state)
- Read current energy consumption and related measurements (voltage, current, power and time stamp of the reading)
- Change device specific settings and configurations such as power measurement resolution or MQTT settings

Once completed, the system should be capable of toggling the relay and relaying the current energy consumption and related data of the Sonoff POW through use of the API endpoint. However, the Pub/Sub model adopted through use of MQTT is preferred over the client-server model of HTTP in IoT connected devices, and as such, the system should now be configured to make use of this.

D. Preliminary Software Experimentation (23-07-2018 to 28-07-2018)

At this point, the hardware should be able to communicate with the stock applications provided by the hardware manufacturers. The remote switch of choice, such as a Sonoff Pow, should be remote controllable over the internet. Additionally, some form of custom firmware should be configured, enabling access over an HTTP API endpoint.

This milestone aims to explore the usage of the custom firmware installed in the previous milstone. A communication protocol should be selected and implemented to control a number of remote switches. Experimentation with communication protocol As discussed before, a communication protocol is required to interact with the Sonoff Pows. MQTT is the chosen protocol for all communication with the device.

Preliminary MQTT experimentation should involve the use of a publically accessible MQTT broker, such as those provided by [10]. For testing this configuration, a local MQTT client is required to be set up on the testing computer. For simplicity, it is recommended to use MQTT-Spy as this comes pre configured with a number of MQTT brokers[11]. The Sonoff Pows should also be configured to access the same MQTT broker.

Controlling the Sonoff Pow via MQTT involves broadcasting a command on a particular topic within the MQTT channel. The commands available can be found on the Tasmota documentation page.

Monitoring Remote power consumption follows the same principle but instead of broadcasting a command to the topic channel one must now subscribe to a particular channel as defined by the Tasmota documentation.

1) Set up local MQTT server: It is ideal to run your own MQTT server, rather than running on a public, open one as all traffic is visible to all members on a the channel. Clearly, this is not ideal as this enables anyone to read and/or control your devices.

To this end, a local MQTT server should be set up. For this project, a Raspberry Pi zero W has been selected as it is low cost and easy to set up. Raspbian Lite is the operating system of choice as it lightweight, has no desktop environment (not required for our purposes) and easy to configure to automatically connect to wifi networks while enabling SSH.

E. Blockchain Related Software Implementation (30-07-2018 to 04-08-2018)

Next, the blockchain smart contract logic for performing accounting and bookkeeping needs to be implemented. In addition, the middleware required to link to the MQTT controlling network needs to be written. Smart contracts will be written in Solidity.

1) Create basic smart contract: The contract model closely follows that of an ERC20 token wherein a mapping is created between an address and an unsigned integer. The address corresponds to the owner's Ethereum address and the unsigned integer corresponds to the individual's balance of tokens. meterBlock implementation varies from the ERC20 token standard by allowing a verified external contract to add/remove tokens from an account. This enables the meter's node to remove tokens on power consumption and add tokens on power creation.

Apon energy consumption, the meter informs the node which intern informs the smart contract of the consumption of power. This results in proportional the deallocation of tokens.

Next, Logic to control switch state based on power consumed is required. To achieve this, the nodes need to monitor the smart contract and when a particular condition is met, such as depletion of an accounts tokens, the load should be switched off.

Finally, one needs the ability to fund existing accounts with new tokens. An interface should be created to facilitate this by creating a website where an account address can be specified to fund with a particular number of tokens.

It is important that any application that deals with the transfer of value, extensive uniting testing should be written to ensure that the application functions as designed. As a result, all functionality should include a suit of tests that will be written by making use of the Truffle Framework.

F. Create Middleware to Connect Blockchain to MQTT server (06-08-2018 to 11-08-2018)

As discussed before, middleware is required to interconnect the MQTT network to the blockchain. This involves tying two different protocols together in one common interface. To achieve this, a Python application must be created using web3.py to interact with the blockchain and an MQTT client to communicate with the Sonoff pows. The MQTT client will read information from the MQTT topic channel, such as power consumed by different households, and report this information periodically to the blockchain through the publication of an Ethereum Smart contract interaction.

At this point in the implementation, one token (henceforth referred to as a Krag token) corresponds to one Kwh. If the user consumers one Kwh of energy, they lose one Krag token from their wallet. This process occurs via the aforementioned communication process.

Once all Krag tokens have been depleted, the node informs the MQTT broker via the middleware that the load should be switched off and the sonoff responds appropriately.

G. Software User interface Implementation (13-08-2018 to 18-08-2018)

A simple user interface is required to provide households with the ability to interact with the application. This application should have a web interface to allow the user to login and view their token balanced, fund their account with new tokens and gain insight into their current usage. This interface should be completely client side without any server side logic, providing access to `web3.js` via Metamask. Ideally, this application should be built using `Vue.js` for simplistic modular design and utilize `Bootstrap4` for consistent styling.

`Metamask` provides the ability to connect a standard web browser, such as Google Chrome, to the Ethereum blockchain via a predefined RPC endpoint. This RPC endpoint will be the Ethereum node hosted on the Raspberry Pi Zero w to simulate the implementation of an Ethereum consortium. Metamask injects `web3.js` (a Javascript library to provide the ability to interact with Ethereum smart contracts) directly into the web browser.

Vuejs is a progressive javascript framework from making scalable, lightweight, modular, production ready applications. Other Frameworks, such as Angular, React or Meteor could be used but Vue is simple and easy to pick up and is recommended for new developers.

H. Intended Use Case Energy Exchange Conceptualisation (20-08-2018 to 25-08-2018)

As discussed from the onset, full use-case application of this system includes scaling the design to be used by all households, businesses, (all producers and consumers) in a national grid. This implementation requires understanding and conceptualisation of the economic forces that control the price per kWh such that the network can dynamically adjust the price based on the current status of the grid.

1) Modelling end user as both consumer and producer: The most general use case of this technology with respect to its final application would be to have one individual household that can either produce or consume energy. A simple input-output model of this can be used to correctly account for the flow of tokens and their associated value based on whether or not the individual is in surplus or deficit of energy(ie. using more energy than generating, or generating more energy than consuming). This abstraction can then be used to model the interaction of these individual households between themselves, and how these tokens are exchanged for power based on current demand.

There are a number of apparent models that can accommodate dynamic price discovera capable of providing a national marketplace for energy exchange. Implementation of these modes are outside of the scope of this project as they are seen as an economic, application level problem. The underpinning technology, built up to this point, can accommodate both variants. However, to fully realise application of this system, these models should be explored.

2) Energy Exchange Modelled as a Traditional Crypto Asset Marketplace: This implementation sees the use of a traditional two sided exchange between a buyer and seller. The current price is defined by the highest buy price and the lowest sell price. The notion of market depth and volume apply in the traditional sense. The buying/selling of energy differs from that of a traditional crypto asset in that there is an additional dimension to consider: time. This means when you sell energy, you need to commit to providing a certain amount of power, for a certain period of time. As a result, energy needs to be packaged into bundles that can then be bought and sold on the exchange. This ensures that the correct amount of power is provided for the mutually defined period of time. An escrow mechanism can be used wherein the seller of energy locks up a proportional amount of tokens equal to the power being sold for the duration of the energy provision contract. In the even that the seller does not hold up her end of the contract (does not provide the correct power for the stipulated period of time), the escrow is used to compensate the buyer.

3) Non-linear bonded Curation Markets: Other applications could involve the use of non-linear bonding curves used to define the relationship between price and volume. This facilitates the creation of a one sided exchange where in you do not require a buyer and seller to exchange power but rather utilize an autonomise price discovery mechanism that leverages predefined commonly agreed upon logic. These bonding curves require complex financial mathematics and are left as an extension to the project research.

V. Conclusion

This report outlined the full scale implementation of a DLT governed device capable of measuring, monitoring and managing the energy consumed or produced by a typical household. Comparisons to the existing models were drawn to provide a justification for the proposed implementation. An extensive, detailed project plan was also provided outlined weekly steps required to achieve the final prototype.

Appendix 1

A. Grid+

Grid+ leverages the Ethereum blockchain to give consumers direct access to wholesale energy markets. In their implementation, Grid+ buys energy wholesale and resells it to the consumer at a lower price than the competition. They use blockchain as the auditing mechanism, to provide the notion of self sovereign ownership over one's power tokens. Their implementation uses a stable coin, backed by the US dollar, known as the BOLT token.

B. Tasmota Firmware Justification

There are three main firmwares to choose from to accommodate the requirements of the project, naimly: Tasmota, Espeasy and Espurna[6], [5], [9], [12], [13]. Each of these can be flashed onto a Sonoff and provide a selection of additional functionality. The primary functionality required is the ability to control the Sonoffs via a HTTP API or over MQTT with a custom broker. All three firmwares offer this functionality and as such the simplest, most lightweight, supported firmware will be used. To this end, Tasmota has been selected as the easiest firmware to achieve the desired functionality.

C. HTTP vs MQTT

HTTP offers the ability to interact directly with the Sonoffs via API endpoints. These API's enable complete control over the device and could work with some form of middleware to enable the Sonoffs to be connected to a blockchain. HTTP works using the notation of a client-server model. This requires the knowledge of the address of the individual sonoffs in order to interact and control them. HTTP is considered to be a document-centric protocol, for transmitting files with a defined start and end, such as a web page over the internet. Streamed data is required to be broken up into packets before transmision.

MQTT on the other hand is considered data centric and uses a client server publish/subscribe pattern. It is designed to be as lightweight as possible and has shown performance improvements of up to 93 times over that of HTTP when used on a 3G network [14].

The publish/subscribe (pub/sub) pattern decouples a client who is sending a messaged (a publisher) from other clients while receiving messages(a subscriber). As a result, the publisher and subscriber do not require to know about each other in communicating over the channel. The last component of the MQTT protocol is called a broker which facilitates the communication between publishers and subscribers. Both the publisher and subscriber need to know about the broker to facilitate this communication. The diagram below outlines the basics of this protocol.

There are a number of advantages associated with the decoupling of a publisher and subscriber, primarily:

- Space decoupling: neither the publisher nor the subscriber need to know about each other. They dont require IP addresses or ports to communicate in the same way that HTTP does.
- Time decoupling: the subscriber does not need to be online for a publisher to transmit a message facilitating temporal decoupling over the communication channel.
- Synchronization decoupling: if the publisher or subscriber is offline, the communication channel does not halt
- Scalability: pub/sub offers far higher scalability than client server due to its ability to break down messages into parallelized components
- Message Filtering: multiple device types can run on the same broker with the subscriber specifying a particular type of message it is interested in, filtering out the rest

D. Select Blockchain of Choice

1) Bitcoin: is the most popular and well known blockchain. It does however have very limited scripting abilities and does not offer a Turing complete programing language capable of providing dynamic autonomous execution of contract code. It's functionality is limited to the transfer of value from one party to another [15]. As a result, if the Bitcoin blockchain was used, heavy modification to the underlying codebase would be required. Moreover, bitcoin can only facilitate three transactions per second, has high transaction fees and extremely slow transaction processing times (up to twelve hours to process a single transaction). As such, payment channels would also be required to be implemented if Bitcoin was to be used. Lastly, Bitcoin uses Proof-of-work as its consensus algorithm making it inherently inefficient at reaching consensus[15]. This ideology is diametrically opposed to that of meterBlock.

2) IOTA: is designed to be more scalable than traditional blockchains, with no theoretical maximum throughput. Additionally, it has no miner fees as there are no miners within the IOTA network. Rather, each node within the network performs the action that a traditional miner would do when processing a transaction. IOTA does not have blocks and as a result, no chain [16]. It is a stream of interlinked transactions that are distributed and stored across the network through a data structure called a Tangle, a form of a DAG. Despite the apparent advantages of IOTA, it does not offer any platform to create smart contracts and as such would require the use of extensive modification to accommodate any form of complex token exchange settlements.

3) Cardano: is a distributed blockchain computing platform. It uses proof-of-stake, making it inherently more efficient than any other proof-of-work alternative [17]. Cardano does plan to offer smart contracts in the future but in its current implementation it does not have this functionality and so it is not applicable for meterBlock.

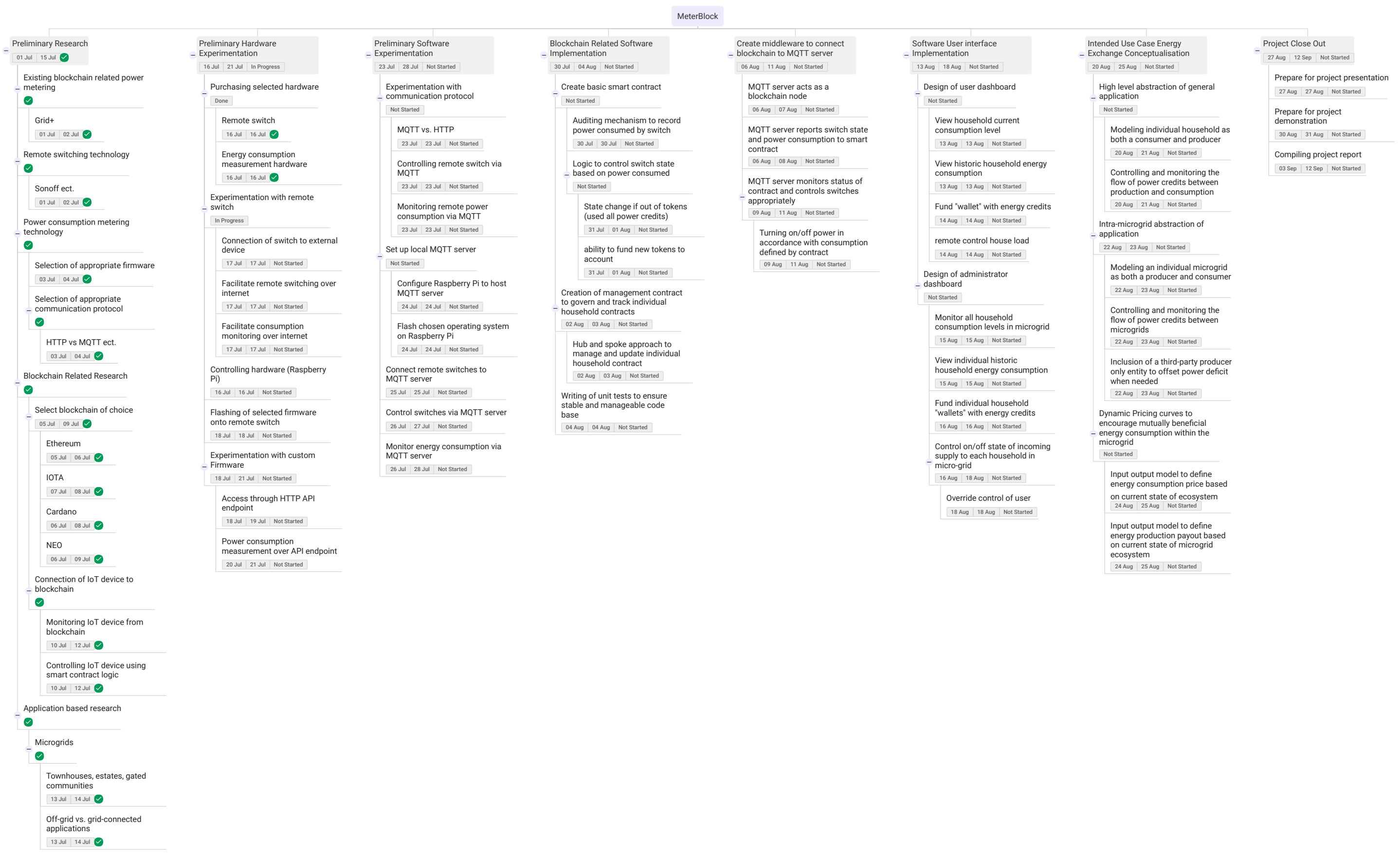
4) NEO: formally known as Antshares, is seen as the Chinese version of Ethereum by many. Fundamentally, it offers much of the same functionality as Ethereum with full smart contract support. Moreover, NEO's smart contract platform is language agnostic meaning that any turing complete programming language can compile down to NEO's smart contract bytecode. NEO uses delegated byzantine fault tolerance (dBFT) as it's consensus algorithm[18]. This makes it efficient and fast but this results in the consensus becoming more centralised than other platforms and as such makes it more susceptible to censorship. Additionally, NEO has a limited English speaking developer base making the interaction with the documentation cumbersome at times. Lastly, NEO's platform is still under development and is not considered production ready.

5) Ethereum: is a blockchain platform offering turing complete smart contracts. It has the biggest developer mindshare by an order of magnitude. Ethereum is currently used in more smart contract related projects than any other blockchain, with 96 out of the top 100 token based applications running on the Ethereum blockchain[16]. Ethereum is currently using Proof-Of-Work, but will soon move to Proof-Of-Stake under the Casper update making it far more scalable and efficient. Ethereum offers the ability to easily run a consortium blockchain between trusted nodes, making the setup and operation of a private blockchain advantageous for testing and development. Lastly, Ethereum has extensive documentation and is currently considered production ready.

REFERENCES

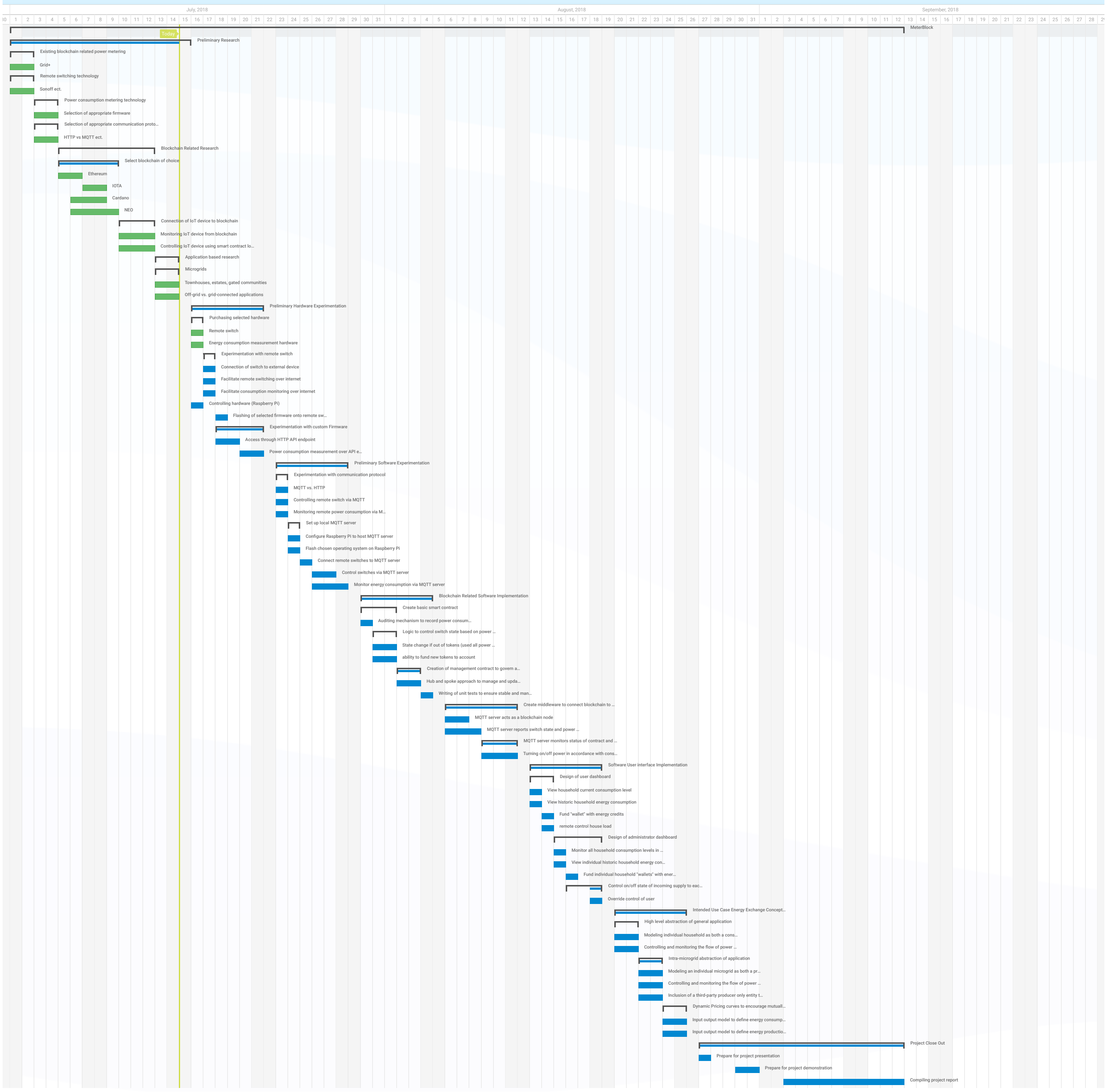
[1] Eskom, "Prepayment History." [Online]. Available: http://www.prepayment.eskom.co.za/history.asp
[2] Gridplus, "Grid+ White paper," 2017. [Online]. Available: https://gridplus.io/assets/Gridwhitepaper.pdf
[3] Solaradvice, "Can I Sell Electricity in South Africa? - Solar Advice." [Online]. Available: https://solaradvice.co.za/can-i-sell-electricity-in-south-africa/
[4] Itead.cc, "Sonoff Pow - ITEAD Wiki," 2018. [Online]. Available: https://www.itead.cc/wiki/Sonoff{_}Pow
[5] Sonoff, "eWeLink Apps on Google Play." [Online]. Available: https://play.google.com/store/apps/details?id=com.coolkit
[6] Mozilla, "HTTP — MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP
[7] HiveMQ, "HiveMQ - Enterprise MQTT Broker." [Online]. Available: https://www.hivemq.com/
[8] MicroRobotics, "Sonoff POW Wifi Switch + PWR Meter - Micro Robotics." [Online]. Available: https://www.robotics.org.za/IM160810001?search=sonoffpow
[9] Arendst, "Sonoff-Tasmota," 2018. [Online]. Available: https://github.com/arendst/Sonoff-Tasmota
[10] HiveMQ, "MQTT Essentials Part 2: Publish & Subscribe." [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe
[11] Kamilfb, "MQTT-Spy," 2016. [Online]. Available: https://github.com/kamilfb/mqtt-spy/wiki/Downloads
[12] Letscontrolit, "ESPEasy - Let's Control It." [Online]. Available: https://www.letscontrolit.com/wiki/index.php/ESPEasy
[13] Xoseperez, "Espurna," 2018. [Online]. Available: https://github.com/xoseperez/espurna
[14] Marina Serozhenko, "MQTT vs. HTTP: which one is the best for IoT? MQTT Buddy Medium." [Online]. Available: https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105
[15] Bitcoin, "Getting Started with Bitcoin Bitcoin.com." [Online]. Available: https://www.bitcoin.com/getting-started
[16] EthereumFoundation, "Ethereum Project," 2018. [Online]. Available: https://www.ethereum.org/
[17] Cardano, "Cardano - Home of the Ada cryptocurrency and technological platform." 2018. [Online]. Available: https://www.cardano.org/en/home/
[18] NEO, "NEO Smart Economy," 2018. [Online]. Available: https://neo.org/

Appendix 2.A



- MeterBlock
  - Preliminary Research — 01 Jul | 15 Jul | ✓
    - Existing blockchain related power metering — ✓
      - Grid+ — 01 Jul | 02 Jul | ✓
    - Remote switching technology — ✓
      - Sonoff ect. — 01 Jul | 02 Jul | ✓
    - Power consumption metering technology — ✓
      - Selection of appropriate firmware — 03 Jul | 04 Jul | ✓
      - Selection of appropriate communication protocol — ✓
        - HTTP vs MQTT ect. — 03 Jul | 04 Jul | ✓
    - Blockchain Related Research — ✓
      - Select blockchain of choice — 05 Jul | 09 Jul | ✓
        - Ethereum — 05 Jul | 06 Jul | ✓
        - IOTA — 07 Jul | 08 Jul | ✓
        - Cardano — 06 Jul | 08 Jul | ✓
        - NEO — 06 Jul | 09 Jul | ✓
      - Connection of IoT device to blockchain — ✓
        - Monitoring IoT device from blockchain — 10 Jul | 12 Jul | ✓
        - Controlling IoT device using smart contract logic — 10 Jul | 12 Jul | ✓
    - Application based research — ✓
      - Microgrids — ✓
        - Townhouses, estates, gated communities — 13 Jul | 14 Jul | ✓
        - Off-grid vs. grid-connected applications — 13 Jul | 14 Jul | ✓
  - Preliminary Hardware Experimentation — 16 Jul | 21 Jul | In Progress
    - Purchasing selected hardware — Done
      - Remote switch — 16 Jul | 16 Jul | ✓
      - Energy consumption measurement hardware — 16 Jul | 16 Jul | ✓
    - Experimentation with remote switch — In Progress
      - Connection of switch to external device — 17 Jul | 17 Jul | Not Started
      - Facilitate remote switching over internet — 17 Jul | 17 Jul | Not Started
      - Facilitate consumption monitoring over internet — 17 Jul | 17 Jul | Not Started
    - Controlling hardware (Raspberry Pi) — 16 Jul | 16 Jul | Not Started
    - Flashing of selected firmware onto remote switch — 18 Jul | 18 Jul | Not Started
    - Experimentation with custom Firmware — 18 Jul | 21 Jul | Not Started
      - Access through HTTP API endpoint — 18 Jul | 19 Jul | Not Started
      - Power consumption measurement over API endpoint — 20 Jul | 21 Jul | Not Started
  - Preliminary Software Experimentation — 23 Jul | 28 Jul | Not Started
    - Experimentation with communication protocol — Not Started
      - MQTT vs. HTTP — 23 Jul | 23 Jul | Not Started
      - Controlling remote switch via MQTT — 23 Jul | 23 Jul | Not Started
      - Monitoring remote power consumption via MQTT — 23 Jul | 23 Jul | Not Started
    - Set up local MQTT server — Not Started
      - Configure Raspberry Pi to host MQTT server — 24 Jul | 24 Jul | Not Started
      - Flash chosen operating system on Raspberry Pi — 24 Jul | 24 Jul | Not Started
    - Connect remote switches to MQTT server — 25 Jul | 25 Jul | Not Started
    - Control switches via MQTT server — 26 Jul | 27 Jul | Not Started
    - Monitor energy consumption via MQTT server — 26 Jul | 28 Jul | Not Started
  - Blockchain Related Software Implementation — 30 Jul | 04 Aug | Not Started
    - Create basic smart contract — Not Started
      - Auditing mechanism to record power consumed by switch — 30 Jul | 30 Jul | Not Started
      - Logic to control switch state based on power consumed — Not Started
        - State change if out of tokens (used all power credits) — 31 Jul | 01 Aug | Not Started
        - ability to fund new tokens to account — 31 Jul | 01 Aug | Not Started
    - Creation of management contract to govern and track individual household contracts — 02 Aug | 03 Aug | Not Started
      - Hub and spoke approach to manage and update individual household contract — 02 Aug | 03 Aug | Not Started
    - Writing of unit tests to ensure stable and manageable code base — 04 Aug | 04 Aug | Not Started
  - Create middleware to connect blockchain to MQTT server — 06 Aug | 11 Aug | Not Started
    - MQTT server acts as a blockchain node — 06 Aug | 07 Aug | Not Started
    - MQTT server reports switch state and power consumption to smart contract — 06 Aug | 08 Aug | Not Started
    - MQTT server monitors status of contract and controls switches appropriately — 09 Aug | 11 Aug | Not Started
      - Turning on/off power in accordance with consumption defined by contract — 09 Aug | 11 Aug | Not Started
  - Software User interface Implementation — 13 Aug | 18 Aug | Not Started
    - Design of user dashboard — Not Started
      - View household current consumption level — 13 Aug | 13 Aug | Not Started
      - View historic household energy consumption — 13 Aug | 13 Aug | Not Started
      - Fund "wallet" with energy credits — 14 Aug | 14 Aug | Not Started
      - remote control house load — 14 Aug | 14 Aug | Not Started
    - Design of administrator dashboard — Not Started
      - Monitor all household consumption levels in microgrid — 15 Aug | 15 Aug | Not Started
      - View individual historic household energy consumption — 15 Aug | 15 Aug | Not Started
      - Fund individual household "wallets" with energy credits — 16 Aug | 16 Aug | Not Started
      - Control on/off state of incoming supply to each household in micro-grid — 16 Aug | 18 Aug | Not Started
        - Override control of user — 18 Aug | 18 Aug | Not Started
  - Intended Use Case Energy Exchange Conceptualisation — 20 Aug | 25 Aug | Not Started
    - High level abstraction of general application — Not Started
      - Modeling individual household as both a consumer and producer — 20 Aug | 21 Aug | Not Started
      - Controlling and monitoring the flow of power credits between production and consumption — 20 Aug | 21 Aug | Not Started
    - Intra-microgrid abstraction of application — 22 Aug | 23 Aug | Not Started
      - Modeling an individual microgrid as both a producer and consumer — 22 Aug | 23 Aug | Not Started
      - Controlling and monitoring the flow of power credits between microgrids — 22 Aug | 23 Aug | Not Started
      - Inclusion of a third-party producer only entity to offset power deficit when needed — 22 Aug | 23 Aug | Not Started
    - Dynamic Pricing curves to encourage mutually beneficial energy consumption within the microgrid — Not Started
      - Input output model to define energy consumption price based on current state of ecosystem — 24 Aug | 25 Aug | Not Started
      - Input output model to define energy production payout based on current state of microgrid ecosystem — 24 Aug | 25 Aug | Not Started
  - Project Close Out — 27 Aug | 12 Sep | Not Started
    - Prepare for project presentation — 27 Aug | 27 Aug | Not Started
    - Prepare for project demonstration — 30 Aug | 31 Aug | Not Started
    - Compiling project report — 03 Sep | 12 Sep | Not Started

Appendix 2.B



July, 2018
August, 2018
September, 2018
Today
MeterBlock
Preliminary Research
Existing blockchain related power metering
Grid+
Remote switching technology
Sonoff ect.
Power consumption metering technology
Selection of appropriate firmware
Selection of appropriate communication proto...
HTTP vs MQTT ect.
Blockchain Related Research
Select blockchain of choice
Ethereum
IOTA
Cardano
NEO
Connection of IoT device to blockchain
Monitoring IoT device from blockchain
Controlling IoT device using smart contract lo...
Application based research
Microgrids
Townhouses, estates, gated communities
Off-grid vs. grid-connected applications
Preliminary Hardware Implementation
Purchasing selected hardware
Remote switch
Energy consumption measurement hardware
Experimentation with remote switch
Connection of switch to external device
Facilitate remote switching over internet
Facilitate consumption monitoring over internet
Controlling hardware (Raspberry Pi)
Flashing of selected firmware onto remote sw...
Experimentation with custom Firmware
Access through HTTP API endpoint
Power consumption measurement over API e...
Preliminary Software Experimentation
Experimentation with communication protocol
MQTT vs. HTTP
Controlling remote switch via MQTT
Monitoring remote power consumption via M...
Set up local MQTT server
Configure Raspberry Pi to host MQTT server
Flash chosen operating system on Raspberry Pi
Connect remote switches to MQTT server
Control switches via MQTT server
Monitor energy consumption via MQTT server
Blockchain Related Software Implementation
Create basic smart contract
Auditing mechanism to record power consum...
Logic to control switch state based on power ...
State change if out of tokens (used all power ...
ability to fund new tokens to account
Creation of management contract to govern a...
Hub and spoke approach to manage and upda...
Writing of unit tests to ensure stable and man...
Create middleware to connect blockchain to ...
MQTT server acts as a blockchain node
MQTT server reports switch state and power ...
MQTT server monitors status of contract and ...
Turning on/off power in accordance with cons...
Software User Interface Implementation
Design of user dashboard
View household current consumption level
View historic household energy consumption
Fund "wallet" with energy credits
remote control house load
Design of administrator dashboard
Monitor all household consumption levels in ...
View individual historic household energy con...
Fund individual household "wallets" with ener...
Control on/off state of incoming supply to eac...
Override control of user
Intended Use Case Energy Exchange Concept...
High level abstraction of general application
Modeling individual household as both a cons...
Controlling and monitoring the flow of power ...
Intra-microgrid abstraction of application
Modeling an individual microgrid as both a pr...
Controlling and monitoring the flow of power ...
Inclusion of a third-party producer only entity t...
Dynamic Pricing curves to encourage mutuall...
Input output model to define energy consump...
Input output model to define energy productio...
Project Close Out
Prepare for project presentation
Prepare for project demonstration
Compiling project report