Investigation of transaction speeds in Blockchain and Tangle technologies

Elliot Jordan

Abstract

Computer science has enabled the possibility of trustless ledgers. A trustless ledger is not under the control of a trusted ‘middle-man’ but instead distributed across a network uncountable nodes that reach a consensus, preventing fraudulent changes. Changes to the ledger requires a degree of work, and each successive change increases the security of the ledger, effectively making it immutable and irreversible. This system is known as Blockchain and it benefits come at a cost; the degree of work causes huge power consumption and restricts the latency of new transactions. The Tangle is proposed to be the next generation of Blockchain, offering practical solutions to these aforementioned issues. Little experiment data exists to settle the matter, therefore there is a need for an investigation into these models. This paper covers the design and implementation of two trustless ledger (Blockchain and Tangle) systems. The systems will be fairly compared using an experimental harness, to derive data on energy and transaction time. This data will be analysed qualitatively so an indication of the best performing model can be given.

Contents

Glossary 3

1. Introduction 4

2. Background 5

3. Literature Survey 11

  3.1 Blockchain 11

  3.2 Tangle 19

  3.3 Comparison and Solution Approach 23

4. Problem Articulation 26

  4.1 Assumptions and Constraints 26

  4.2 Problem Statement 27

5. Design and solution approach 28

  5.1 Solution Approach 28

  5.2 Model design 31

  5.3 Design of Blockchain 45

  5.4 Design of the Tangle 53

6. Implementation 60

  6.1 Graphical User Interface 60

  6.2 Common Code 71

  6.3 Blockchain Implementation 75

  6.4 Tangle Implementation 80

  6.5 Experimental Harness 85

  6.6 Tests 86

7. The Experiment 96

  7.1 Aims 96

  7.2 Method 97

  7.3 Results 98

  7.4 Limits of validity 100

8. Discussion and Conclusion 102

  8.1 Results Analysis 102

  8.2 Comparison and Discussion 103

  8.3 Social, Ethical, Legal and Health and Safety Issues 104

  8.4 Conclusion and Future Work 104

9. Reflection 107

  9.1 On the Project 107

  9.2 Personal 107

References 109

Appendices 116

Glossary

Confirm – Confirming a transaction implies that the currency is sent, and the receiver can spend it.

Crypto – abbreviation for cryptocurrency

DAG – Directed Acyclic Graph

Mining – informal verb to describe Proof of Work

POW – Proof of Work

TPS – Transactions per Second

TPM – Transactions per minute

RSA – Rivest–Shamir–Adleman (asymmetric encryption algorithm)

ECDSA - Elliptic Curve Digital Signature Algorithm

Merchant confirm – A transaction is considered valid and confirmed by a merchant after it has been added to the Blockchain and has been followed by ***x*** subsequent valid blocks.

1. Introduction

Throughout history, ledgers have been used by banks, merchants and governments to create a reliable transcript of transactions. During this time ledgers have evolved, to increase profitability and utility, and eventually became electronic. Traditionally they are under control of a trusted organisation or person. Computer science has enabled the concept of a trustless ledger; a ledger that is not owned or controlled by any one organisation. In this new regime the ledger is held on the consensus of an uncountable number of nodes distributed across the internet. The complexity of effecting fraudulent change guarantees the immutability of the ledger. However, an undesirable side effect is the potentially excessive demand on computing resources and thus energy consumption. There are two major approaches, Blockchain and Tangle, and these are compared in this study.

Both approaches abide by the fundamentals of a trustless ledger; an uncountable number of nodes, immutability and irreversibility. Their approach differs when it comes to the way transactions are stored on the ledger, and the underlying protocols involved in adding transactions. IOTA, a Tangle implementation; claims the Tangle is the next generation of Blockchain due to its lesser energy consumption and fast transaction process time. These claims are vaguely demonstrated by existing performance data but lack any experimental evidence in a fair environment. The aim of this report is to test these claims experimentally; in regard to energy consumption per transaction, and the time it takes for a transaction to be considered valid.

To experimentally test these variables, a solution is required which can replicate the functionality of the Tangle and Blockchain. This is achieved by implementing two systems; one being a full Tangle implementation, and the other a full Blockchain implementation. Both of these systems can be ran independently as a trustless ledger but can also be utilised in an experimental harness. This experimental harness initialises multiple nodes of the respective trustless ledger to simulate a network of nodes. Enabling the testing of both ledgers in a fair controlled environment where independent variables can be customised.

This report comprehensively establishes the problem at hand then details the design of the solution. It begins by covering the background of ledgers, then directs the literature to focus on the current problem domain. The full research, design, and implementation of both trustless ledgers is detailed in accordance with the solution approach. The implementations are tested in respect to the prior mentioned variables, and the results are analysed and discussed.

2. Background

Ledgers and money in history

Ledgers have been used for banking and record keeping since 3200 BC by the ancient Mesopotamians [1]. They used clay tablets to record quantities. They divided the tablets into rows and columns with each cell of the tablet detailing an item and quantity. The item was marked using the earliest discovered form of human writing 'Proto-cuneiform' [2], which uses pictures to represent words. The quantity of the item was marked by pushing holes into the tablet. The use of these tablets is well documented [3] and included their use as receipts for orders. Each tablet would normally consist of a transaction, its purpose, calculations of values, the signature of an authorising official, an authenticating cylinder seal and a date.

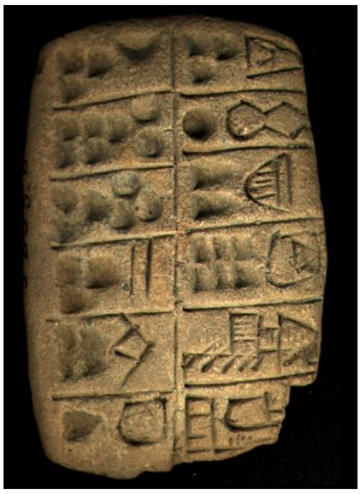


Figure 1 - *An ancient tablet denoting multiple types of grain products and their quantities. The two bottom right cells denote the responsible official and the receiving official.* [3]

In 1300 AD there was a revolution of sorts regarding ledgers. Medieval Europe was gradually moving to a monetary economy [4] and a need for bookkeeping became intrinsic, especially amongst merchants. A new form of bookkeeping, double-entry bookkeeping, was invented which in turn moulded the shape of the economy. During a time when loans were common and necessary for many, double-entry bookkeeping was a system used to keep track of debtors and creditors. Transactions would be listed down twice: as a credit and as a debit [5]. This enabled banks and lenders to optimise their capital and minimise their liability, somewhat leading the way for the power banks hold today.

Paper money first emerged in 800 AD in China [6]. The Silk Road was a network of hugely profitable trade routes that China took full advantage of. However, transporting valuable metal coins was inefficient as long distances needed to be covered. Instead, banknotes were used that were redeemable for metal coins in the Chinese capital. By 1100 AD paper money had become fully acknowledged and authenticated by the Chinese government [7]. Paper money was widely circulated and as a result the government outlawed unofficial banknotes and added counterfeiting measures to their notes.

Paper money during this time could be considered commodity money, i.e. money that is backed by a physical good, normally gold or silver [8]. The definition of commodity money does also stretch to allow a commodity such as fur to be a currency. Essentially an individual could exchange their paper money and debts for this physical good [9] at a bank or government exchange. An advantage of commodity money was that it allowed governments to control the flow of money being released, to match inflation via the gradual increase in the value of gold.

Modern Implementations

Fiat money is tender which is made legal by government decree. It is not backed by any physical good or currency and its value is derived from trust in the distributing government. With no intrinsic value it must be viewed as a promise that the money can be exchanged for its value in goods [10]. Although first appearing centuries beforehand, fiat money became popularised and successful in the 20th century. This can be somewhat apportioned to the events of 1971, when the US President at the time cancelled the direct convertibility of the US dollar to gold.

In addition to having no intrinsic value, in recent years money has become more and more electronic. An individual can now see their bank balance online, including incoming and outgoing transactions, much like a ledger. Paper money is slowly being phased out, as contactless and card payments dominate the market share [11].

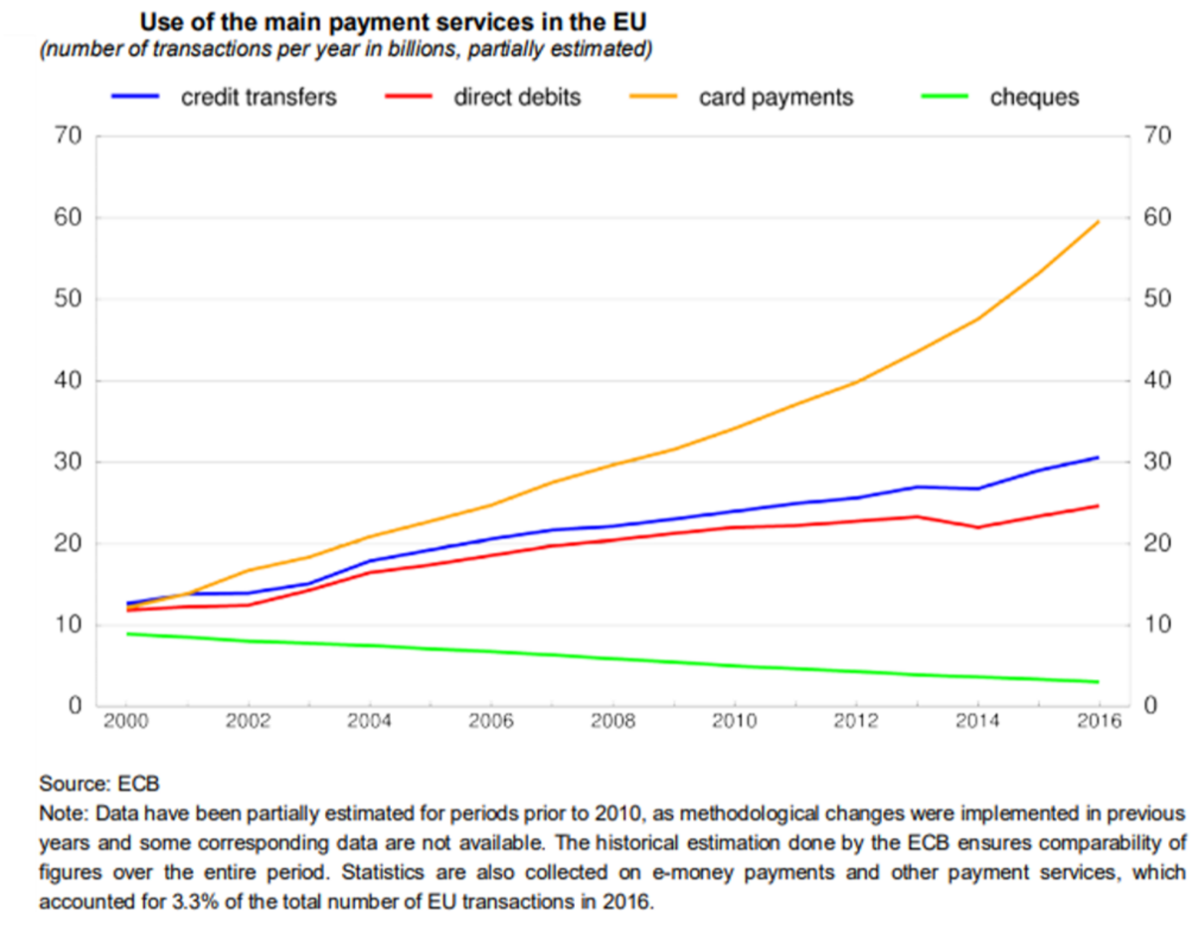


Figure 2 - *Clearly viewable trend of card payments rising in the billions from year to year, while cheques decline.* [12]

Electronic money is still fiat money, requiring trust that it has any value. However, storing most of your money in banks adds an additional layer of risk. Banks are only required to physically store ~10% of their deposits [13] and the remaining money is lent to other people and organisations. When trust in banks is low or people fear banks will run out of money, bank runs can occur [14]. Causing a bank to run out of physical cash can result in huge wait times or being unable to withdraw money. It is unlikely an induvial will lose their money if a bank goes bust as it is insured. However, if the total deposits of an individual goes over a certain threshold (£85000 in the UK) [15], the excess will be lost.

In addition to trusting banks, scepticism has risen over the safety of electronic payments. Fraud is a primary concern, with losses of £731.8 million in 2017 via unauthorised financial fraud across payment cards, remote banking and cheques [16]. Contactless payment is somewhat flawed by design, as the cost of fast transactions comes at the risk of unauthorised transactions. Losses are mitigated by allowing daily caps on cards and a limit of £30 per transaction, but the danger of fraud still lingers.

Trustless Ledgers

In October 2008 Satoshi Nakamoto published a White Paper for a peer-to-peer electronic cash system: Bitcoin [17], the first trustless ledger system. It was proposed as an alternative to making electronic transactions through financial institutions. The Paper was published a month after Lehman Brothers filed for bankruptcy [18], which propelled the ongoing financial crisis into the 'worst financial crisis in global history' [19]. This crash led to the public losing a lot of trust in banks, with 66% of people in the UK still not trusting banks to work in the interest of the UK as of August 2018 [20].

Satoshi Nakamoto set out a vision for a digital currency that would not need to use banks as a third-party, effectively creating the first trustless currency in the form of a ledger. In Satoshi's implementation the name of the currency shared the same name as the software: Bitcoin.

Bitcoin uses a platform which Satoshi called blockchain, to achieve trustless, immutable and irreversible transactions [21]. Consisting of a series of securely connected blocks with transactions within, a blockchain should include every transaction ever made on a ledger and be publicly viewable by all. Users can host a node on the Bitcoin network where they can 'mine' Bitcoins by doing a large amount of CPU work, also known as proof of work. These nodes communicate new transactions and blocks to each other to create a consensus. This consensus is required for changes to be made and therefore for the ledger to function at all.

The value of a Bitcoin is backed by the work done to create a coin and the long chain of previously done work on the blockchain. The ledger cannot be changed unless all proof of work on every block is redone, which is largely unfeasible. When considering the recent financial crisis, trusting fiat money could be a risk that bitcoin and blockchain can avert.

Blockchain was the first implementation of a trustless ledger, but by no means the last. Since 2008 the cryptocurrency ecosystem has flourished, with a significant amount of research and many new cryptocurrencies. The cumulative market cap of cryptocurrencies started at around $1 billion in 2013 and is now $146 billion (as of March 2019), peaking at a total of just over $800 billion in 2017 [22]. Bitcoin has had the highest market cap of all other currencies throughout this time, mainly due to its status as the legacy cryptocurrency and being somewhat considered the 'gold standard' [23] of crypto.

The computer science-based technologies of trustless ledgers

Trustless ledgers rely heavily on three fundamental technologies of computer science: peer-2-peer nodes, asymmetric cryptography and databases. For a trustless ledger to be trustless it must be verifiable by all participants, while giving no one total control, and to satisfy these requirements it must be distributed.

Peer-2-Peer is a networking architecture that forms a network of equally privileged and powerful nodes. These nodes can share computing resources to work on mining coins and communicate new transactions and other updates.

Asymmetric cryptography uses two mathematically identical keys to create a public key and a private key. Much like a username and password, the public key can be considered a user's pseudonym that can be freely shared. The private key must be kept secret otherwise funds will be at risk. When a transaction is sent to the blockchain it must be validated with the sender's private key. To do this the transaction is 'signed' with the private key, such that the key cannot be derived from the signature. The signature however can be verified as being signed with the correct private key by using the public key of the user. Without this protocol there would be no way of validating transactions, making a distributed ledger unachievable.

A database can be simply defined as 'a structured set of data held in a computer'. For a trustless ledger to be a ledger it must store every transaction ever made. The type of data structure is not important, but the ledger must contain within each transaction at a minimum: from address, receiver address, an amount, timestamp and a signature. Depending on the implementation more information may be required.

These fundamentals define a trustless ledger as a peer-2-peer distributed database, which holds public-key encrypted money of pseudonymous participants. Implementations of trustless ledgers should closely follow these fundamentals. Without them they cannot be considered trustless or perhaps even a ledger. Controversy surrounding the governance of ledgers often comes into focus as owners of cryptocurrencies make suspicious changes to their code or ledger [24].

Strengths and weaknesses of trustless ledgers

The significant aims of trustless ledgers should be to combat the issues found in traditional ledgers but also to be able to replicate their benefits. In theory this can be achieved. The positives of trustless ledgers are listed as follows:

- Trustless
- Immutable
- Irreversible
- Fast
- Fraud-proof
- No personal data on ledger (only public key)

The benefits are convincing and present a strong case for cryptocurrency adoption; however, in practice flaws begin to seep through. Trustless ledgers have not been immune to criticism and are often accused of having suboptimal implementations, which lead to problems with issues that cryptocurrencies were explicitly meant to solve. There are also fundamental flaws shared across all cryptocurrencies by the nature of being a crypto.

The largest problem – and the basis of this project and investigation – is transaction speeds in the blockchain network. In Satoshi Nakamoto's White Paper he specified that blockchain should add one block to the chain every 10 minutes. A single block can only contain 2000 transactions, meaning that if there are too many transactions waiting to be added to the block they can be waiting indefinitely. At one time Bitcoin had over 200,000 transactions waiting to be confirmed [25].

By design, blockchain nodes compete against each other to mine coins first. As only one block can be mined every 10 minutes, then as more nodes are added to the network, more work is cumulatively done per block mined. The secondary focus of this investigation is the energy used per transaction. In large blockchain networks the amount of energy used can be unbelievably huge. In December 2018 Bitcoin used 496 KWh per transaction [26], which would cost about £61 using UK energy prices [27].

The Tangle is new implementation of a trustless ledger. It claims to be an implementation that is not susceptible to high energy and time costs. By using a Directed Acyclic Graph (DAG) instead of a linear chain of blocks along with additional changes, transactions can take substantially less time in comparison with blockchain.

Little experimental data exists explicitly comparing the two implementations. This leaves a space to assess if these claims are reputable and if the Tangle is perhaps the next generation of Blockchain.

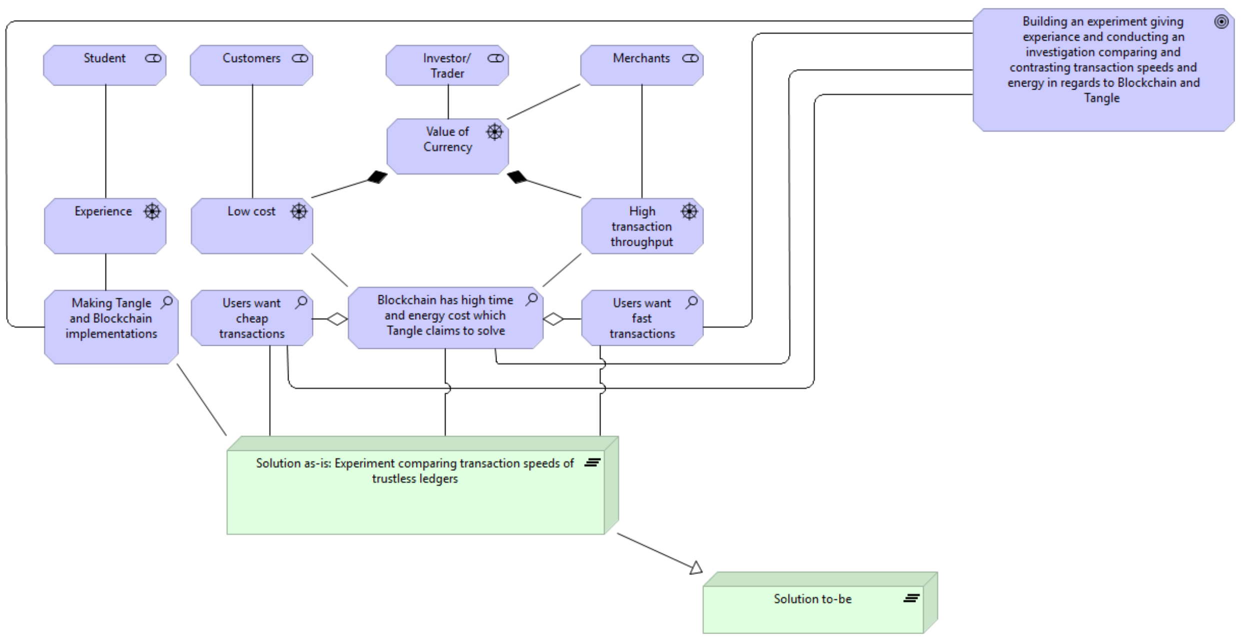
Inspiration for project

In early 2018, during my placement year, I spent time learning the basics of blockchain. As I pursued this interest I began to delve deeper into the technology behind blockchain. I used tutorials to learn how to replicate different parts of the blockchain in code, and eventually became aware through the media of the negatives of Blockchain and the obstacles it faced. I also became aware of the Tangle, an implementation of a trustless ledger that supposedly fixed the problems of the Blockchain model. After familiarising myself with the Tangle, I realised there was a lack of experimental data explicitly comparing the two. There also appeared to be a need for this data, as the cryptocurrency community is split on the best implementation. I felt I had enough knowledge both to pursue an investigation into the Tangle and Blockchain, and to meet the advanced development requirements.

Understanding Problem Domain and Plateau

Trustless ledgers appear to be the future, assuming they have a legitimate use-case over electronic money. Many implementations of the trustless ledger exist, so finding the right implementation is key. Blockchain is the main candidate, but the Tangle claims to solve the high time and energy problems associated with Blockchain. Verifying this claim is a primary goal of this report.

The premise for a trustless ledger has been defined above. Using the definition of a trustless ledger and the problems already mentioned, a basic problem articulation can be composed. This model (which will be improved upon later in the report) sets the premise for how the project will be approached.



Student
Customers
Investor/ Trader
Merchants
Building an experiment giving experiance and conducting an investigation comparing and contrasting transaction speeds and energy in regards to Blockchain and Tangle
Value of Currency
Experience
Low cost
High transaction throughput
Making Tangle and Blockchain implementations
Users want cheap transactions
Blockchain has high time and energy cost which Tangle claims to solve
Users want fast transactions
Solution as-is: Experiment comparing transaction speeds of trustless ledgers
Solution to-be

Figure 3 - *Preceding analysis for Goal Sketching to derive basis of initial problem statement*

Summarised outline of the structure of report

The report begins with a literature survey on Blockchain and Tangle functionality and their problems. This leads onto a problem articulation for an experimental investigation on the transaction speeds of blockchain and tangle. A design section follows, describing the experimental design and discussing the constraints and assumptions in detail. An implementation section then explains the implementation of the adequately representative models of Blockchain and Tangle. An experiment is performed using the implementations and the results are reported and discussed. Attention is given to the limits and validity of the results, possible future work and the conclusions. The final sections of the report reflect on the project and its results.

3. Literature Survey

The scope of this Literature Survey is on finding the necessary information to create a suitable experimental model of both Blockchain and Tangle. Furthermore, problems existing in both models will be analysed to justify the need for the experiment.

3.1 Blockchain

Blockchain and Bitcoin have been introduced above, along with their fundamental computer science technologies. However, it is in the combination of these technologies and their subtleties that the ingenuity of Blockchain is found. To implement a Blockchain solution for the experiment, the actual Blockchain solution needs to be reviewed first. This section of the report will break down Blockchain into parts, gathering the information required from the Bitcoin developer documentation guide [28]. Additionally, the problems present in Blockchain will be specified, with the intention of comparing it to Tangle, and the need for experimentation justified.

3.1.1 How Blockchain works

Block Structure

As mentioned earlier, Blockchain consists of a linear structure of connected blocks. In Bitcoin the data structure of a single block is large and hard to follow [29], partly as it has metadata which is unnecessary for the sake of experimentation. The Ethereum White Paper [30] details a block structure that can be simplified into the following elements:

- Index/Height of the block – Position of the block within the Blockchain, increments by 1 each block.
- Timestamp – Reference point for when the block was created; would expect that blocks would ascend in chronological order along the chain.
- Hash – Hash of all the other information included in a block, normally has ***x*** number of zeros at start of hash to correspond to difficulty.
- Hash of previous block – Connects latest block to last block on chain; anything else would not be valid.
- Nonce – Number only used once; see Proof of work for more details.
- Extra Nonce – Extra Number only used once; also explained in Proof of work.
- Difficulty – The difficulty of the Proof of work; corresponds to the resulting hash of the block.
- Reward – number of coins rewarded to address of miner.
- Merkle Root – generalisation of the hash list of transactions in the block.
- List of ***x*** transactions – Blocks include an amount of transactions which are data structures themselves.

The first block in the Blockchain is known as the Genesis block [31]. Its values are normally fixed and defined in the code. The rest of the chain will then be built using the Genesis block as the starting block. Any other chain that starts with a block that isn’t identical to the Genesis block provided will be invalid.

Blockchain and other trustless ledgers rely on hash functions to keep them trustless. A hash function is a function that can map any data of any arbitrary size to a hash of a fixed size [32]. An important feature of hash functions is that it is unfeasible to derive the inputs of a function from the hash. These functions perform a role in mapping the data from blocks and transactions into fixed length strings. In turn, blocks and transactions can be easily verified with hashes and Proof of work is possible using hash functions. A slight change in the detailed inputs of a hash function will result in a hugely different resulting hash, making it impossible to cheat the system. An important implication of hash functions is that they are collision resistant, meaning ‘it is computationally infeasible to find any two distinct input x, x’ which hash to the same output’ [33]. If this was not the case someone could maliciously attempt to create a different block with the same hash.

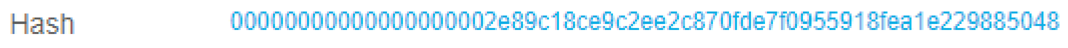

Hash 0000000000000000002e89c18ce9c2ee2c870fde7f0955918fea1e229885048

Figure 4 - *The hash of a Bitcoin Block* [34]

This makes the hash one of the most important parts of a block, which is calculated in the Proof of work algorithm. All information in the block is hashed together making a fixed sized hash, meaning that if any of the contents are changed without recomputing the hash, then other nodes will be able to tell it has been modified and reject it. Recomputing the hash is not an easy task as the Proof of Work done to create the initial hash must be repeated. It is also not worthwhile as all information within the block will also be independently verified to make sure it is also valid.

Each block includes the hash of the previous block in its contents too. This previous hash is included in the Proof of Work algorithm to make the current hash. This effectively creates a chain of immutable blocks as each block will refer to the previous block in the chain [35]. Changing the previous hash in a block means the block’s actual hash must be recomputed.

This chain plays a vital role in making Blockchain immutable and irreversible. For example, if a rogue node wished to make a change in block 100 in a chain of 1000 blocks, it would have to correct the previous hash and recompute the Proof of work for the next 900 blocks in the chain. This amount of work is completely unfeasible and will take a substantial amount of time, even with a lot of computing power [28]. In addition to this the Blockchain network of nodes would not accept these changed blocks. This is because nodes only append to the blockchain and do not swap/change blocks. There is only one condition in which a Blockchain node will make changes to old blocks, but it involves having a valid Blockchain which is longer than the current chain the node is working with. This is explained in more detail in the Forks section.

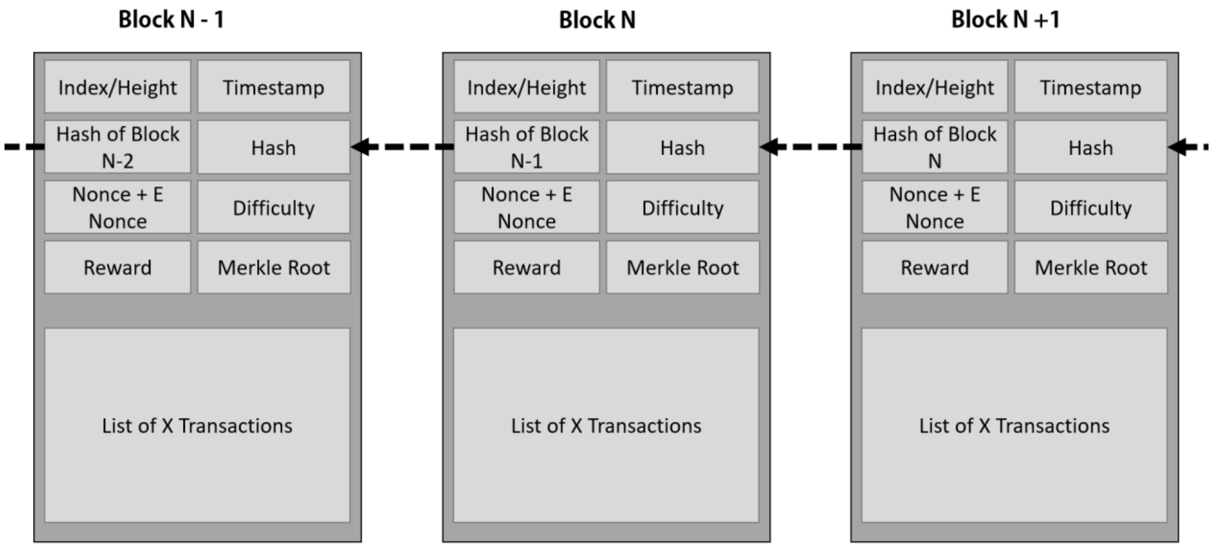


Figure 5 - *Visualisation of the data structure of a block*

The Merkle Root is a vital component of Blockchain and verifies the integrity of transactions within a block. The Merkle Root is the result of a Merkle Tree function, where the hashes of each transaction are hashed together to make one overall hash. In Blockchain, every transaction in a block is paired with another; the hashes of the transaction pairs are then hashed together to make a single hash for each pair. This process is then repeated, each hash is paired with another and they are hashed together until only one hash remains, which is the Merkle Root [36].

Merkle Roots confirm that identical information exists across two different locations, as the Roots should both be the same. This is useful for a node receiving a block, to confirm that all transactions were included and unchanged. This allows blockchains around the world to quickly and efficiently coordinate records across computers [37].

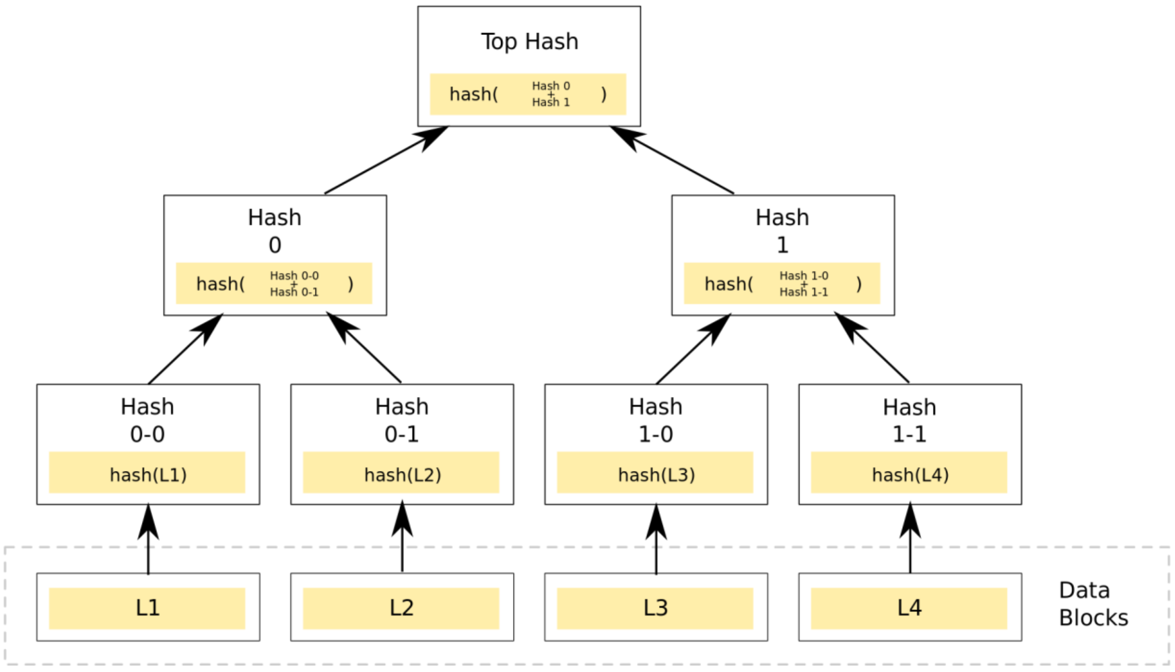


Figure 6 - *Visual Representation of Merkle Tree function, where Data Blocks represent the hash of each transaction* [38]. *The Merkle Root would be the top hash in the tree.*

Transaction Structure

Each time a user or node wishes to spend their currency, they will need to generate a transaction and send it to a node. Once a transaction is sent to a node it is propagated across the peer-2-peer network [28]. Depending on the size of the fee and traffic of the network it will be eventually picked up and added to a future block.

When a user makes a transaction, they will need to specify: their public key, their private key, an amount and a fee. This information is then hashed, and the hash is then signed with the private key to make a signature [28]. Given that 'keys are correlated cryptographic numbers which can be added or multiplied' [39] it is possible to mathematically confirm that the transaction is valid by confirming the signature using the sender's public key. As mentioned previously this is achievable via asymmetric encryption.

Fees in Blockchain are a small amount of currency sent on top of the original amount, also known as gas in Ethereum [30]. This fee is given to the address of the miner who mines the block the transaction was included in. In a network of 1000s of other transactions, a miner would be more inclined to be 'selfish' and pick up all the transactions with a high fee. Therefore, to get a transaction confirmed quickly a higher fee helps. As of April 2018, the cheapest and fastest transaction fee for Bitcoin is 148 satoshis (0.00000001 Bitcoins/satoshi) per Byte [40].

As a result, the structure of a transaction is as follows:

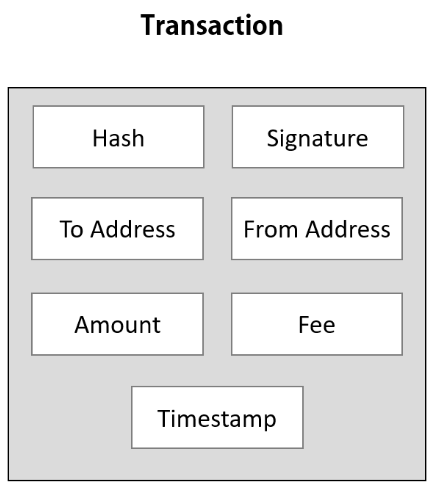


Figure 7 - *Data structure of a transaction in Blockchain*

The Bitcoin White Paper briefly mentions in Section 8 payment verification [17]. It claims a user can see if their transaction has been confirmed by checking that ‘blocks added after it further confirm the network has accepted it’ after the initial block has been added. In practice this is also how transactions are confirmed. Instead of immediately confirming a transaction after it has been added to the chain, ***x*** number of blocks must be added after it until it is considered confirmed. In Coinbase, a cryptocurrency exchange, this number is 6 for Bitcoin and 35 for Ethereum [41]. This is worthy of consideration for the experiment, as in practice a transaction will need to be confirmed 6 times, which could have a measurable effect on the overall transaction time.

Proof of work

Proof of Work is a mechanism originally devised to counter the problem of masses of spam emails being sent during the 2000s [42]. Proof of Work was originally proposed to require email senders to perform a resource-intensive computation before sending an email. An extract from ‘Proof of Work can Work’ claims ‘The intention is to deter spam by making it uneconomic to send a large number of messages, while enabling legitimate users to send small numbers of messages’ [42].

In Satoshi Nakamoto’s White Paper [17], Proof of Work is repurposed for adding blocks to the blockchain, proposed to stop blocks being spammed on the chain and prevent attacks on the network. Before a block can be added to the chain its Proof of Work must be completed first. This work involves computing the hash of a block until a hash is found that starts with ***x*** number of zeros. As hash functions will always give the same output with the same input, the input to the hash function should change in every attempt of the work. As it is important all the data in the block is valid and unchanged, most of it cannot be changed. Therefore, an additional value is used in the hash function, a nonce.

A nonce is a number only used once; after each attempt to hash the block the nonce is incremented. This means a different input is used each time and therefore a completely different output can be expected. Eventually after many tries the resulting hash will fulfil the objective of finding a hash with ***x*** number of zeros at the start. This will probably be a substantial amount of work which will take a long time to complete. In Bitcoin blocks are designed to be mined once every 10 minutes [28], meaning it should take 10 minutes for the entire network to mine one block.

The number of zeros required in the block hash is dictated by the difficulty variable. In Ethereum difficulty is defined as ‘A scalar value corresponding to the difficulty level of this block. This can be calculated from the previous block’s difficulty level and the timestamp’ [30]. Bitcoin dynamically changes the difficulty after each block is mined to keep the block time as close to 10 minutes as possible. Therefore, an increase in difficulty will take longer to mine, and more mining power will be required on the network to mine a block in 10 minutes. This mechanism is common across blockchain but configured differently for each currency. For example, Ethereum aims for a much lower block time, which can be measured to a value of roughly 15 seconds [43].

Repeated work is a topic of concern in blockchain. Hypothetically if two nodes were attempting to mine an identical block and both started the nonce at 0 and incremented up, they would both be repeating each other’s work. To prevent this an extra nonce is used. A random extra nonce is calculated for each node mining. This extra nonce should never be the same amongst any two nodes. By including this extra nonce in the hash function no two inputs should ever be repeated, resulting in no repeated work. An additional benefit of the extra nonce appears when the nonce has incremented to its max value. A 32-bit integer nonce can only increment to 2.14 billion so after this many hash attempts it is out of space to work with. Consequently, the extra nonce can be incremented and the nonce reset allowing the POW algorithm to continue [44].

Given the large computational requirements in mining a block it needs to be worthwhile for a node to mine. Therefore, when a block is mined the mining node’s address is rewarded with a fixed amount of currency. This amount is normally set at the developer’s discretion. In Bitcoin rewards are distributed with the intention to only ever distribute a finite amount of coins. This is known as controlled supply [45]. Originally mining a block rewarded the node with 50 Bitcoins. Every 210,000 blocks that are mined this reward is reduced by 50%. This will continue until no more Bitcoins are mined. Overall roughly 21 million coins will be mined, resulting in a fixed supply.

Nodes want to mine as fast as possible to be the first block to finish, to get the block reward. Lots of hardware can be used to push the most hashing power out of a node as possible. Asymmetric threading is a useful component to utilise as much hashing power as possible. This is because multiple threads can hash independently of each other and hash functions have a low overhead per use [46].

It should be mentioned that this is not the only protocol for adding blocks to the chain. Many other protocols have been designed for this purpose. Some of them are used in blockchain implementations, such as Proof of Stake. While it is not necessary to describe them in this Literature Review, they will be reflected on in the Conclusion.

Block Validity

Blockchain’s immutability and irreversibility originates from the many validation and verification methods built into it. Blocks can be validated by recalculating the provided hash and checking the previous hash. The transactions in blocks can be validated by confirming the signature and checking if the sending address has enough currency for the transaction. In addition, the Merkle Root can be recalculated to ensure all transactions are included. Other checks include validation that the difficultly, reward and hash match the expected number.

Forks

When nodes on a Blockchain network have different chains from each other this is known as a fork. This is an unworkable situation as different chains will have different transactions on them, meaning it is impossible to tell which transactions are confirmed. To fix this, Blockchain always gives priority to the longest chain. Other nodes will fix their chain, finding the point where the chain forked and deleting all blocks past that point. After that the longer chain will be added to the point. This is known as a soft fork [47] as it can be fixed. In theory this creates the potential for double spends on the network, as a transaction could be mined in a block and then overridden by a larger chain when it has been confirmed.

A hard fork is defined as ‘a change to a protocol that renders older versions invalid’ [48]. This normally involves changing parameters that would be rejected by the old protocol, such as block size or difficulty algorithm. When disagreements in the way to run a Blockchain implementation occur, hard forks can follow. This is made evident by the Paper ‘On the Security and Performance of Proof of Work Blockchains’ [49], which says that ‘Bitcoin has been forked a number of times in order to fine-tune the consensus’.

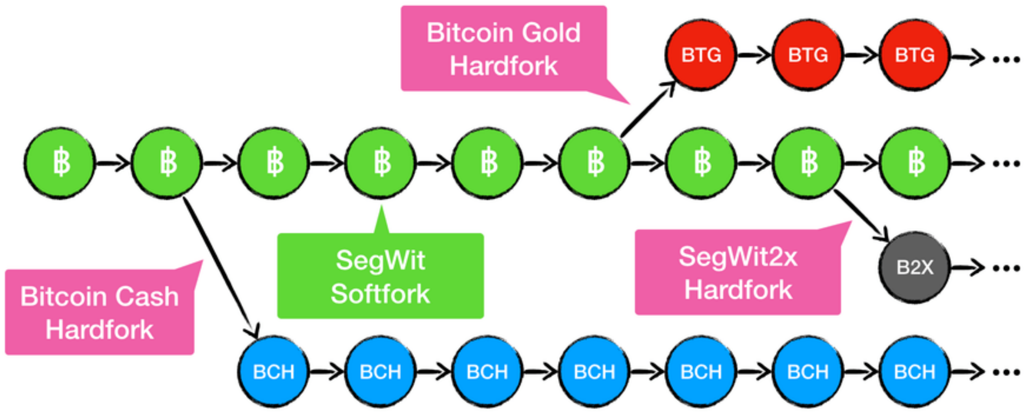


Figure 8 - *Notable Hard Bitcoin forks up to 2017* [50]

Transactions and Blocks in the Network

The network is a set of processes, N, called nodes that are fully connected. Nodes can send messages to other nodes that operate synchronously. Nodes can read messages and if necessary will broadcast a message to all other nodes in response.

In the Bitcoin White Paper [17] the network was summarised in 6 steps:

1. New transactions are broadcast to all nodes.

2. Each node collects new transactions into a block.

3. Each node works on finding a difficult proof-of-work for its block.

4. When a node finds a proof-of-work, it broadcasts the block to all nodes.

5. Nodes accept the block only if all transactions in it are valid and not already spent.

6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

An important distinction not mentioned in this summary is how transactions are stored while they are waiting to be added to a block. When new transactions are sent to a node they are stored in its memory pool. If the node chooses not to include the transaction in the next block, it will stay there. However, if the memory becomes full it may choose to drop some transactions with low fees [51]. While it is unlikely a transaction will ever be fully lost across the network, the possibility for nodes to drop it exists, and this could have some bearing on transaction confirm time.

3.1.2 Blockchain Problem Breakdown

As Blockchain is configured to only add a new block to the chain every ***x*** number of seconds, this is the minimum amount of time a transaction can get processed in. Compounding this with the fact that multiple confirms are required by brokers [41], the true sum of time taken can be impractically large for true adoption. This does not even include the possibility that multiple blocks can be mined before the transaction in mind gets processed. It is normal to expect a 60-minute wait before a transaction will be confirmed in Bitcoin [52]. In comparison with electronic money, contactless and card payments are virtually instant, and an hour wait will not usually be tolerated. Although electronic payments normally take multiple days to appear on a bank statement, protocols are in place to accept the payment anyway.

2000 transactions per block poses a scalability problem for Bitcoin in the future. At 10 minutes per block, Bitcoin can only handle a maximum of ~3.33 transactions per second (TPS). This problem is present in many implementations of blockchain and many attempts at a solution exist. Other implementations such as Ethereum can handle 20 TPS but ultimately, for world-wide adoption, a system needs to handle 1700 TPS [53].

Multiple confirms are required by brokers as they need to be confident the chain will not get overridden and the transaction deleted. A 'double spend' can be defined as a flaw in a trustless ledger that allows the same digital token to be spent more than once. When nodes pick up transactions to mine, they ensure these transactions do not already exist on the chain. If a transaction is confirmed and then overridden this can also be considered a 'double spend' [54], as the transaction can be added back to the chain again.

The most devasting type of double spend attack is known as a 51% attack. This is when a node or nodes that controls over 50% of the current computation power being used to maintain a Blockchain currency has the intention to double spend by overriding the Blockchain. A hidden fork of the chain will be mined while the rest of the network builds the main chain. The rogue nodes will then submit transactions, which in turn will be added to the main chain. After these transactions are confirmed (once ***x*** number of confirms have been completed) the longer hidden fork will be shared across the entire network. As the nodes are programmed to always select the longest chain, the blocks with the rogue transactions will be overridden by the hidden fork, resulting in a successful double spend. The risk of this happening on Bitcoin is unlikely as 50% of the computational power is almost an unfeasible amount to reproduce. However, there was an incident in Bitcoin Cash where a mining pool had over 50% of the total computational power and could potentially have caused a 51% attack [55].

The high energy throughput involved in Proof of Work brings scepticism to the entire Blockchain model. Proof of Work was originally a system designed to prevent spam and double spend attacks in Blockchain. In Bitcoin, the total computational power is estimated to be a minimum of 22 TWh, almost the same as Ireland [56]; other sources put this number at 50 TWh [26]. However, this remains profitable for the miners, as mining a single block brings a reward of 12.5 bitcoins, and at $5000 [57] each as of April 2019, a block is worth $62500 not including transaction fees. Subsequently, this means the Bitcoin network can afford to spend $62500 in energy per block mined, implying that if the Bitcoin price remains high then extortionate amounts of energy will continue to be used.

Trustless ledgers are inherently public to all, meaning that as every transaction ever made can be seen, an individual's address can be tracked by anyone and their spending behaviour analysed. This can be used for good and has been used to track down users/distributors of illegal dark web sites via their addresses [58]. Nevertheless, it remains an ethical grey area and may discourage adoption in the future. Monero is a Blockchain implementation which proposed in its White Paper [59] a protocol to make transactions and addresses anonymous.

In a finite currency Blockchain implementation, after rewards have stopped being distributed nodes will still need an incentive to continue mining. The fee system will have to be the sole income for miners. In theory this should cause inflation, as the fees of the currency will have to at least match the cost of the energy to mine the block. However, this could also lead to huge fees on transactions to get them confirmed. In Bitcoin it currently costs £61 in energy per transaction processed [27]. This cost is subsidised by the 12.5 bitcoins rewarded for mining. Without that reward, each transaction will have to pay such an energy cost at a minimum in fees [60].

3.2 Tangle

The Tangle was proposed in 2018, in the paper 'The Tangle' [61] and it is implemented in a cryptocurrency called IOTA. The paper proposes that the Tangle is the next evolutionary step of the blockchain by offering features to make a full micropayment system. The Tangle sticks to many of the fundamentals of trustless legers: a network of nodes to reach consensus, public and private keys for transactions and the database-like structure of transactions. Blocks and Blockchains are not utilised, in preference for a directed acyclic graph.

3.2.1 How Tangle works

Directed Acyclic Graph

A directed acyclic graph is a directed graph that does not have any cycles. Tangle uses a directed acyclic graph to store transactions instead of blocks in a blockchain. Transactions are stored as vertices on the Tangle, and the edges represent the confirmation of one transaction by another.

When a transaction is added to the Tangle it must select two other transactions already on the Tangle to confirm. It should then validate these two transactions plus all transitive predecessor transactions of them [62]. The selection of these two transactions should favour the tips/unconfirmed transactions of the Tangle but doesn't need to. This process results in new transactions confirming the other recent transactions in an ongoing process. To confirm a transaction a small amount of work must be done, but this is still exponentially less work than Blockchain [61].

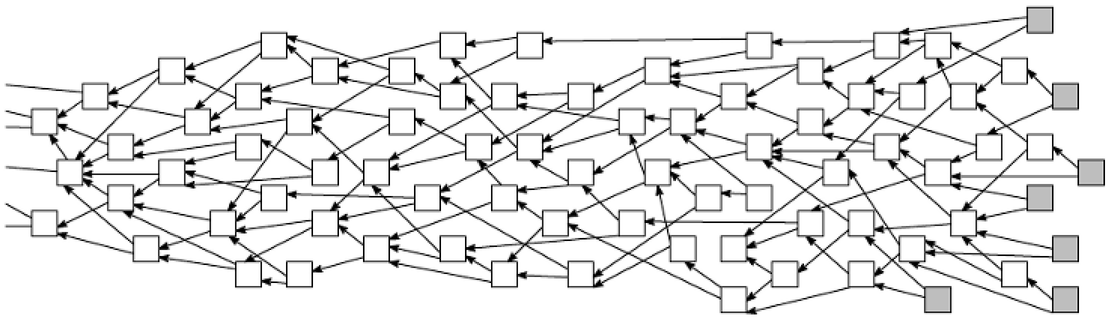


Figure 9 - *Illustration of a Directed Acyclic Graph* [63], *tips of the graph are coloured grey. Directed edges show the confirmation of one transaction by another.*

Nodes hosting a Tangle can have different Tangles to each other and transactions can be added in different orders. Ideally this variance should be small as large differences could make Tangles incompatible with each other.

If eventually combined, running a temporary offline fork/subtangle of the Tangle is possible. A subtangle can combine with the online Tangle later, if it would be legal and valid to combine them. The process of creating a subtangle is documented below.

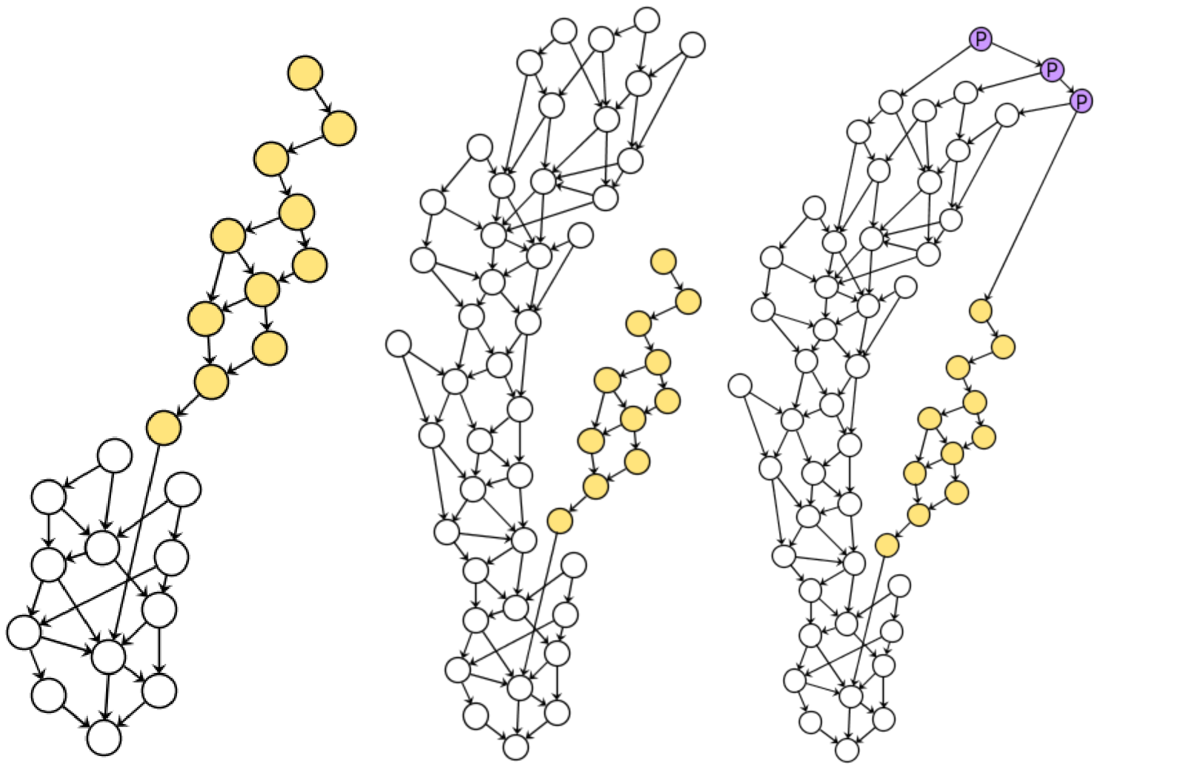


Figure 10 - *An offline Tangle is created, highlighted in yellow. This subtangle originates from a transaction from the whole Tangle. To recombine with the whole Tangle transactions, it needs to be broadcast to the node network. The Tangle will then begin to confirm transactions on the subtangle.* [64]

Transactions

Transactions largely follow the same structure as a Blockchain transaction with a few subtle differences. There are no fees but instead a weight. Weight is proportional to the amount of work that the issuing node invested into it [61]. This value is adjusted dynamically by the network to prevent spam. There is also a height attribute: instead of incrementing after a transaction is added, it is calculated using the highest height of the two confirmed transactions +1. Multiple transactions can exist on the same height but no transactions with the same height will exist on a path through the Tangle.

$H_N = max(H_{N-x}, H_{N-y}) + 1$

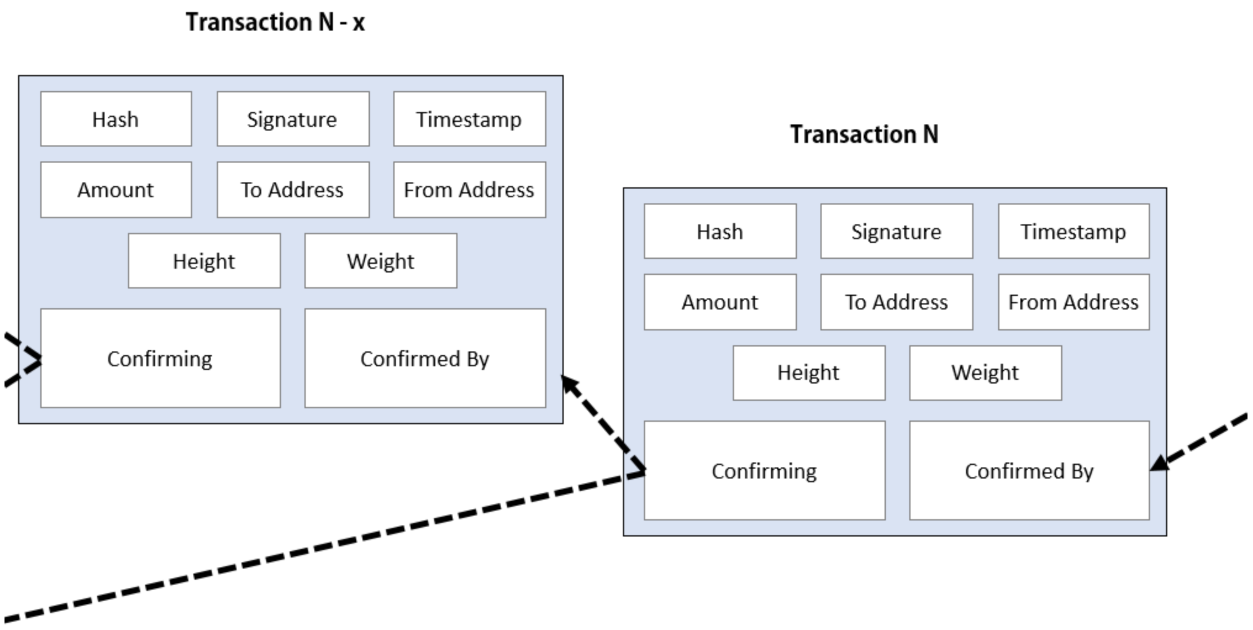


Figure 11 -*The header of two Tangle transactions N and N-x, where N has confirmed N-x. 'Confirming' and 'Confirmed by' are sub-data structures.*

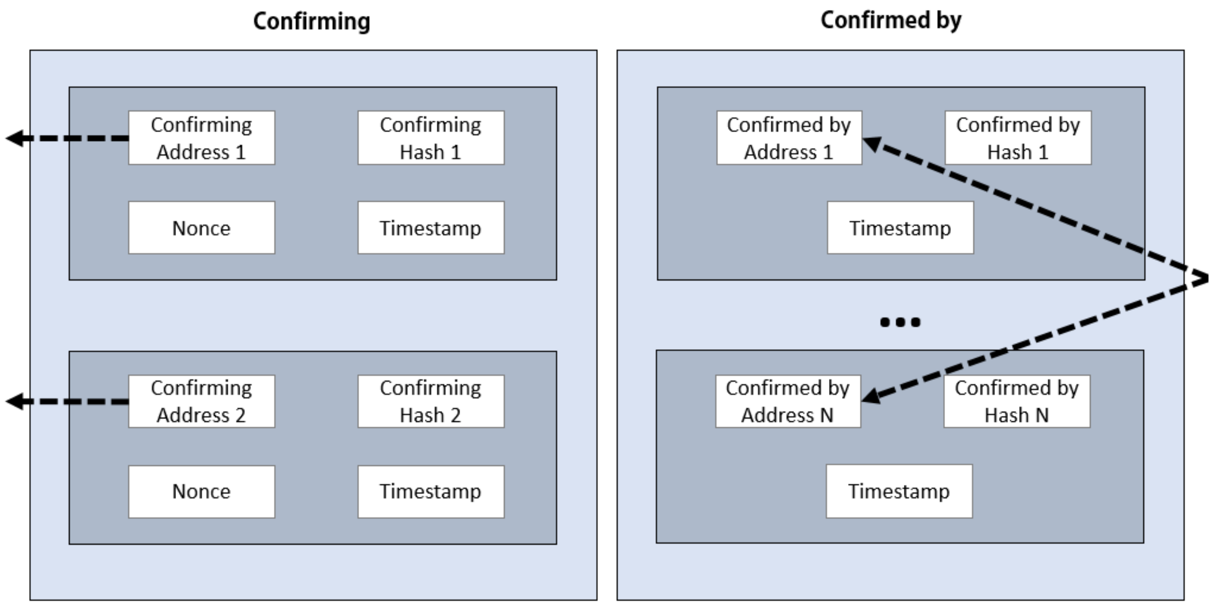


Figure 12 - *Sub-data Structures: 'Confirming' infers the transactions that have been confirmed in this transaction; 'Confirmed by' infers the transactions that have confirmed this transaction.*

Tip Selection

Ideally a new transaction should select unconfirmed transactions to confirm. This results in a network where new transactions are confirmed as fast as possible and the height of the Tangle is increased, making it cryptographically harder to attack. Selecting the correct tips involves choosing the right algorithm.

The Markov chain Monte Carlo is recommended by IOTA given its useful probability distribution. The Paper ‘The Stability and Security of the Tangle’ [65] explains the algorithm in the context of the Tangle. The Markov chain Monte Carlo starts by placing a fixed number of walkers on the local DAG. The local DAG will only be a collection of the ***x*** most recent transactions as travelling through the entire DAG would be too computationally intensive. Each walker performs a random walk towards the tips, calculating the probability of choosing the right path at each vertex, finding multiple tips with no confirms and creating probability densities. The probability of a tip being chosen can be expressed as

$$P_{xy} = \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)\left(\sum_{z:z\rightsquigarrow x} \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_z)\big)\right)^{-1}.$$

[61]

Proof of work

Before a transaction is added it must complete the Proof of work for two other transactions on the Tangle. This work confirms those transactions and creates a connection in the form of a directed acyclic graph that can be reproduced without a graph data structure. The Proof of Work is like Blockchain as it uses an incrementing nonce. Yet the difficultly of the Proof of Work is much lower as defined by the weight. As no other nodes will know of a transaction until it has already completed its work, there is no competition between nodes. This ensures no repeated work or inflated difficulty.

There are no rewards for successfully adding a transaction to the Tangle. As the POW is so easy, transactions can be added to the chain rapidly; it would be unreasonable to reward a node. There are no fees either, and due to the relative ease of the work it is reasonable to expect a user or node to do its own work. With no rewards or fees, no currency can naturally exist in the network. IOTA’s solution was to pre-mine the currency and distribute it via crowdfunding [66].

Confirm Logic

New unconfirmed tips are more likely to get picked by tip selection than old tips. If an old transaction has not been confirmed it can be rebroadcasted to the network to encourage nodes to pick it. This normally occurs when an unconfirmed tip is removed from a node’s local DAG.

Tangle transactions do not need multiple confirms before vendors accept them as valid, unlike Blockchain. This is because once a transaction is confirmed it must be valid and it cannot be deleted or overridden.

The Tangle hypothetically excels in high usage, as the more transactions that are added to the Tangle the more transactions that are being confirmed. With 2 confirms per transaction it is likely that in a high usage network a tip can be confirmed in seconds.

IOTA Specific Implementations

IOTA uses additional mechanisms to confirm transactions and store them. In IOTA a milestone transaction is distributed by a trusted Coordinator node every few minutes to the Tangle. Any transactions connected or indirectly connected to the milestone are considered confirmed. However unconnected transactions are not confirmed until connected [67].

Snapshots are used by IOTA to take an image of the tangle. All transactions are removed from the Tangle and the balances of all addresses are added up. The Tangle is then started again from this snapshot, resulting in a smaller tangle meaning nodes use less storage.

3.2.2 Tangle Problem Breakdown

While Tangle should excel when under high demand, in low demand the network may remain very stagnant. Without an incentive to run a node, the network will have to rely on users creating transactions. If no one is creating transactions, tips will stay unconfirmed indefinitely. Furthermore, there appear to be security concerns for a low usage network. An attacker could spam the network and potentially take control of it and successfully double spend if another node doesn't notice.

The Tangle has a slight dependency on trusted nodes and centralisation. In current implementations it uses a controller and trusted nodes to confirm transactions. Not only does this stall transaction times but it also takes away from the fundamental vision of trustless ledgers being trustless [17]. Its dependency on centralisation significantly secures the network, and a secure implementation of a Tangle in its current form may be reliant on some centralisation measures.

Pre-mining the currency breeds many problems itself. Without a way to earn the currency no one will want to run a node. Selling the currency via crowdfunding goes against the example set by Bitcoin and damages the integrity of the currency from the start. It is also possible that the creator could choose to hold onto a substantial share of the currency, with the possibility that they could act in bad faith, dump their coins and subsequently crash the market.

3.3 Comparison and Solution Approach

3.3.1 Overarching Crypto Problems

The value of cryptocurrency has fluctuated wildly since its inception. The idea of its full adoption and usage in real world scenarios receives a large amount of criticism [68]. Without instant confirmation protocols and a lack of reference point for price, cryptocurrency appears to be relegated to investing and expenditure on the dark web [69].

A trust epidemic has occurred across the crypto community. With so many implementations of trustless ledgers, people are beginning to realise that many of these ledgers are dishonest, with intent to scam customers or remain centralised. This distrust is spreading over every cryptocurrency, resulting in cynicism on all fronts [70].

The legality of cryptocurrency is a contentious issue. Many countries are in the process of regulating cryptocurrency. At G20 crypto was discussed, and they decided to 'implement the Financial Action Task Force standards', although it is unclear what this means [71]. Some countries have prohibited it outright. China has implicitly banned cryptos and prohibited financial firms from holding, trading or profiting from cryptocurrencies [72].

3.3.2 Existing performance measures of Tangle and Blockchain implementations

Using pre-existing data to compare Tangle and Blockchain implementations, indicative results can be obtained. The data come with some assumptions and constraints, which are noted below.

Power Used

Trustless Ledger Type (Implementation)	Power used per transaction
Blockchain (Bitcoin)	496 KWh [26]
Tangle (IOTA)	0.00005 KWh [73]
Blockchain (Ethereum)	39 KWh [74]

Current Transactions per Second*

Trustless Ledger Type (Implementation)	Current Average TPS
Blockchain (Bitcoin)	5.4 [75]
Tangle (IOTA)	6.3 [76]
Blockchain (Ethereum)	6.6 [77]

**Measuring average transactions per second being added to network*

Max Transactions per Second*

Trustless Ledger Type (Implementation)	Max TPS
Blockchain (Bitcoin)	8.68** [78]
Tangle (IOTA)	Theoretically infinite [79]
Blockchain (Ethereum)	25.3 [80]

**Maximum amount of transactions that can be processed per second*

***With Segwit, only 3.3 TPS with 1MB blocksize*

Nodes on network

Trustless Ledger Type (Implementation)	Current nodes on network
Blockchain (Bitcoin)	9669 [81]
Tangle (IOTA)	Data not available
Blockchain (Ethereum)	8329 [82]

The websites providing this data can only record nodes that gave permission for their data to be shared. Therefore, this number may not be truly indicative of the real network size.

Node Hashes on network

Trustless Ledger Type (Implementation)	Hash Power*
Blockchain (Bitcoin)	50 TH/s [83]
Tangle (IOTA)	Data not available
Blockchain (Ethereum)	126.5 TH/s [84]

*Where 1 TH/s is 1 trillion hashes per second.

3.3.3 Why there is a need for experimentation

Performance data does exist – as shown above – although there is a noticeable absence of experimental data for fair comparison. The current transactions per second data shows little difference between Blockchain and Tangle in terms of transactions received. The lack of IOTA transactions hardly proves the Tangle's claim of infinite scalability, leaving both the Tangle and Blockchain on even ground. The power used per transaction is significantly different, massively favouring the Tangle. (It should be mentioned that the only available source of data on IOTA power consumption is a tweet via Twitter, so there are question marks over its reliability.)

The data, however, may be subject to bias. Companies providing the data may stand to benefit from damaging or promoting certain cryptocurrencies. While the Blockchain raw data is reasonably trustworthy, it is difficult to find IOTA data from sources that do not appear slightly biased.

With little experimental data comparing the performance of Blockchain and Tangle fairly, there appears to be a need for a definitive answer. The cryptocurrency industry has reached over $800 billion in market cap [22] but the issue is still contested. An agreed approach forward could help progress and solve divisions within the crypto community.

4. Problem Articulation

The possibility of trustless ledgers is likely to have disruptive influence on business. Rather than needing a trusted middleman (e.g. bank) to keep a ledger honest it is now possible to use distributed storage and computation across the internet to maintain immutable ledgers. The success of this depends on information being held on countless nodes on the internet and requires a significant amount of computation to make additions to ledgers. There are two principal technologies worthy of comparison: Blockchain and Tangle. The former is more established and underpins Bitcoin and Ethereum's service. Tangle is relatively new and underpins IOTA, a new but strongly performing service.

Each technology has its own pros and cons. For example, the energy cost of many blockchain-based currencies and services is reckoned to be untenable. On the other hand, the centralisation that is somewhat present on IOTA undermines its trustless aspect. There is a need to evaluate and compare these technologies. As the literature survey shows, there is little empirical data to prove their claims, and few generally understood educational models and explanations of the principles.

It is proposed that representative instances of each of the two approaches are to be constructed and used as a way of introducing and comparing the principles of operation, and as an initial testbed for comparing computation issues (such as efficiency, implications on scale). The centralisation of the Tangle models can be bypassed in development to give a full comparison of performance in an honest testing environment. The testbed will need to consider both forms of transaction confirmation: merchant confirmation [41] and technical confirmation.

The data gathered on the current implementations of Blockchain and Tangle should be scaled to finite reproducible figures, achievable with limited hardware and processing power. Nodes hosted, and the uncountable power of their networks must be scaled, but meaningful results still generated. Given the huge code bases of each of the system implementations, the development should be scoped to focus on the detail drawn from the literature survey, as this is a final year project.

Due to the practical limitations of the resources available, the testbed would only require experimental prototypes that are good enough to inform the requirements for future development and experiments. The results of the experimental prototypes should be considered qualitive data.

4.1 Assumptions and Constraints

An indicative problem statement should list assumptions and constraints. These are identified here.

Assumptions:

- An uncountable number of nodes can be represented as a finite amount on a single computer.
- Nodes for Blockchain systems can have their network hashing power reduced to the power of a single computer and operate similarly to a full-scale network.
- The security of both implementations need not be considered too much, as the system will be honest. (Comprehensive validation methods will still exist.)

Constraints:

- Blockchain and Tangle systems can run in a virtualised environment on a single computer.
- Limited timeframe to produce experimental prototypes.

4.2 Problem Statement

Blockchain and Tangle models are reported to have significant measurable differences between them. In particular, scalability (the amount of transactions processed per second) and efficiency (the amount of energy used per transaction). With little empirical data there is a need to evaluate and compare these variables while working within the scope of a final year project.

Subject to assuming that an uncountable network of nodes and its hashing power can be finitely represented on a single computer, an experimental testbed can be developed, and tests performed to extract this data. This testbed will consist of full implementations of both Blockchain and Tangle models. The tests performed should satisfy the objectives, these being two experimental prototypes that report energy used per transaction, and time per transaction. The testbed should consider the two definitions of transaction confirmation: technical and merchant. An additional objective is the ambition of gaining experience in developing trustless ledgers. Achieving these objectives should be done in accordance with the constraints on time and the limitations of a single computer.

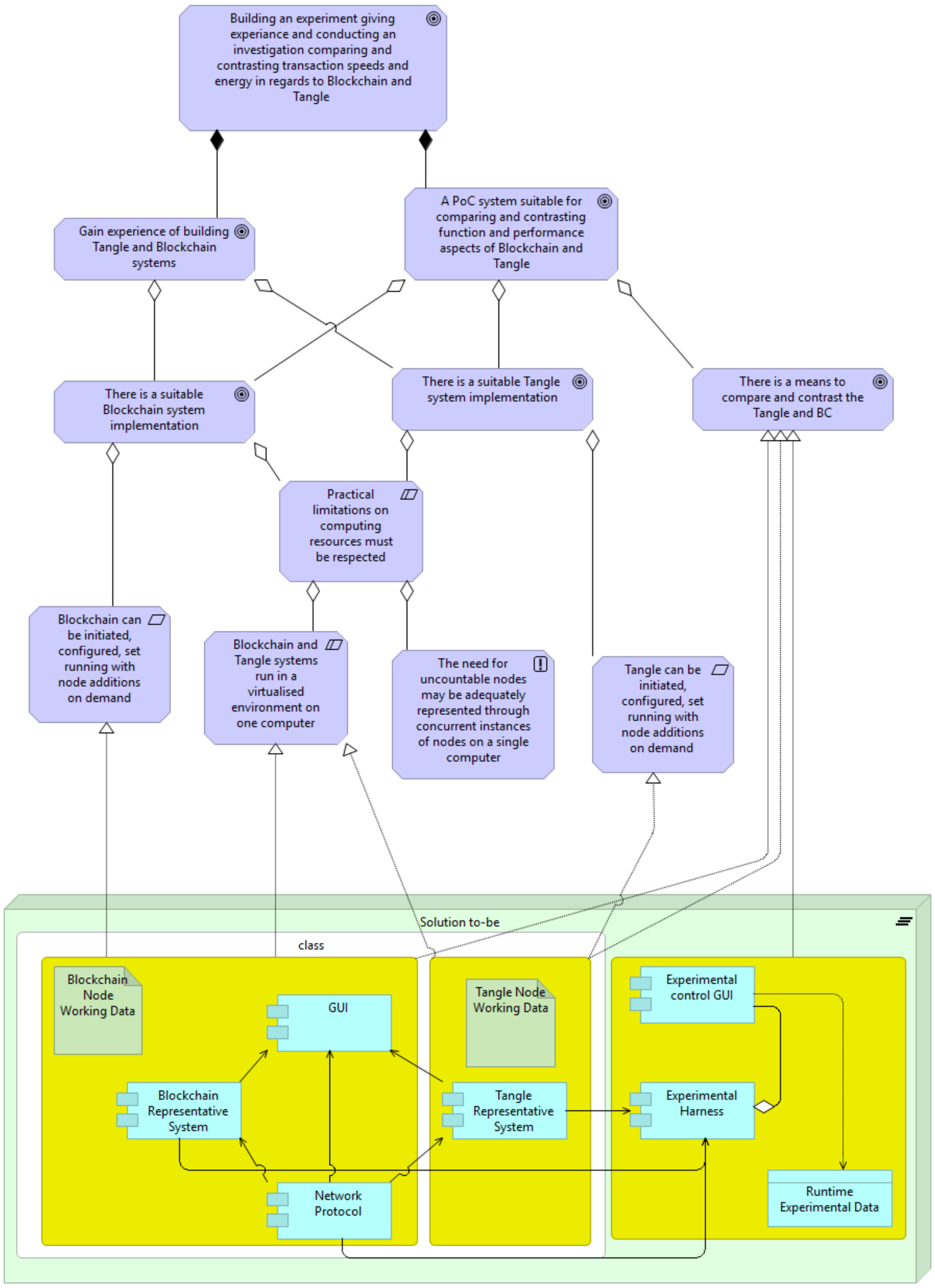
5. Design and solution approach

5.1 Solution Approach

The literature survey has evaluated different implementations of trustless ledgers and the components used to make each one. By understanding their problems, a problem statement has been derived which requires the development of an experimental testbed. This testbed will require a full Blockchain and Tangle implementation for usage on a local PC. When designing the solution approach, it will be important to ensure the systems are as complete as possible, in comparison with a real implementation.

Representative instances of both Tangle and Blockchain should be constructed and used to introduce and compare their principles of operation. These implementations can also be used as an initial testbed for comparing the issues in focus: transaction speed/scalability and energy/work done per transaction. Given the limitations in resources for a final year project, the testbed would need to be experimental prototypes that are good enough to inform the requirements on future development and experiments. The results should be expected to be qualitative as opposed to quantitative. There is also an element of personal ambition in the project, to gain experience in Blockchain and Tangle.

Using the preceding analysis to derive the basis of the initial problem statement, the goal of the investigation has been found using the assessments in the analysis. This motivation can be broken down using Goal Sketching to produce a solution approach. The requirements, constraints and assumptions that have been isolated in the problem articulation are used in the Goal Sketching to design a system which can be validated.



Building an experiment giving experiance and conducting an investigation comparing and contrasting transaction speeds and energy in regards to Blockchain and Tangle
Gain experience of building Tangle and Blockchain systems
A PoC system suitable for comparing and contrasting function and performance aspects of Blockchain and Tangle
There is a suitable Blockchain system implementation
There is a suitable Tangle system implementation
There is a means to compare and contrast the Tangle and BC
Practical limitations on computing resources must be respected
Blockchain can be initiated, configured, set running with node additions on demand
Blockchain and Tangle systems run in a virtualised environment on one computer
The need for uncountable nodes may be adequately represented through concurrent instances of nodes on a single computer
Tangle can be initiated, configured, set running with node additions on demand
Solution to-be
class
Blockchain Node Working Data
GUI
Blockchain Representative System
Network Protocol
Tangle Node Working Data
Tangle Representative System
Experimental control GUI
Experimental Harness
Runtime Experimental Data

Figure 13 - *Solution Approach*

Software Development Tool

Using the correct development tool to implement the system is important. Bitcoin uses C++ [28], Ethereum is built from several languages and IOTA is built using Java, C++, Rust and Go [85]. A language that offers built in functionality for cryptography and networking will be preferable. C# provides an object orientated paradigm suitable for representing data structures such as transactions and blocks. These objects can be stored inside a ledger object for easy indexing, making object orientation a convenient utility for designing the system. C# also has built in asymmetric cryptography, hashing, networking and threading libraries. This should make C# a suitable tool to develop the entire system on, as it minimises the amount of work required by providing existing code.

Component Breakdown

The solution approach model has listed the components required. These components will all require a reasonable amount of engineering, and their functionality and requirements can be broken into their parts. This breakdown will be done in accordance with the literature survey and existing implementations, such that each system is designed as meticulously to a real implementation as possible.

Common code used between both Representative systems:

- Wallet generation – public and private keys
- Transaction signing and signature verification protocol

Blockchain Representative System:

- Block Data Structure
- Blockchain Transaction Data Structure
- Blockchain Data Structure
- Reading and writing ledger from file mechanism
- Mining configuration tool
- Block preparation including threaded Proof of work algorithm
- Transaction generation
- Difficulty Algorithm
- Transaction Pool
- Validation Mechanism
- Merkle Root Algorithm

Tangle Representative System:

- Tangle Transaction Data Structure
- Tangle Data Structure applying DAG logic
- Reading and writing ledger from file mechanism
- Tip Selection algorithm
- Transaction preparation including double Proof of Work algorithm
- Validation Mechanism

Network Protocol:

- TCP Listener thread
- TCP client send thread
- Connection Data Structure
- Connection Pool of IPs

- Message Parser
- Message Preparer

Experimental Harness:

- Configurable Node Controller
- Mechanisms to control and manipulate multiple node controllers
- Analysis mechanisms

GUIs functionality will be broken down in the design section.

Chosen Solution

The technical details of trustless ledgers analysed in the literature survey will be used to design and implement the components. The requirements broken down earlier will be developed within their respective components. C# is the programming language selected to develop this solution.

Based on the objectives and approach it is necessary to build:

- An Experimental Blockchain implementation
- An Experimental Tangle implementation
- An Experimental Harness and data visualisation

This solution approach should satisfy the goals and objectives set in the problem statement. The components of the solution enable experimentation, and therefore experiment generation of the data primarily required. The harness is designed in such a way that transaction data can be collected and analysed from all experimental nodes. The implementation is complex and a huge scope, which should satisfy the personal requirement to learn about developing these systems.

5.2 Model design

The requirements and components have been identified along with the development language, C#. Now these components need to be put together in accordance with the requirements to design functioning models and wireframes that can be understood and implemented. Good performance will be a desirable trait of the system as the experimental results are time sensitive. A strong design should help circumvent any flaws from the start.

The solution to-be contains a rudimentary UML class diagram of the expected solution. This is more visible in the Basis of detailed requirements diagram.

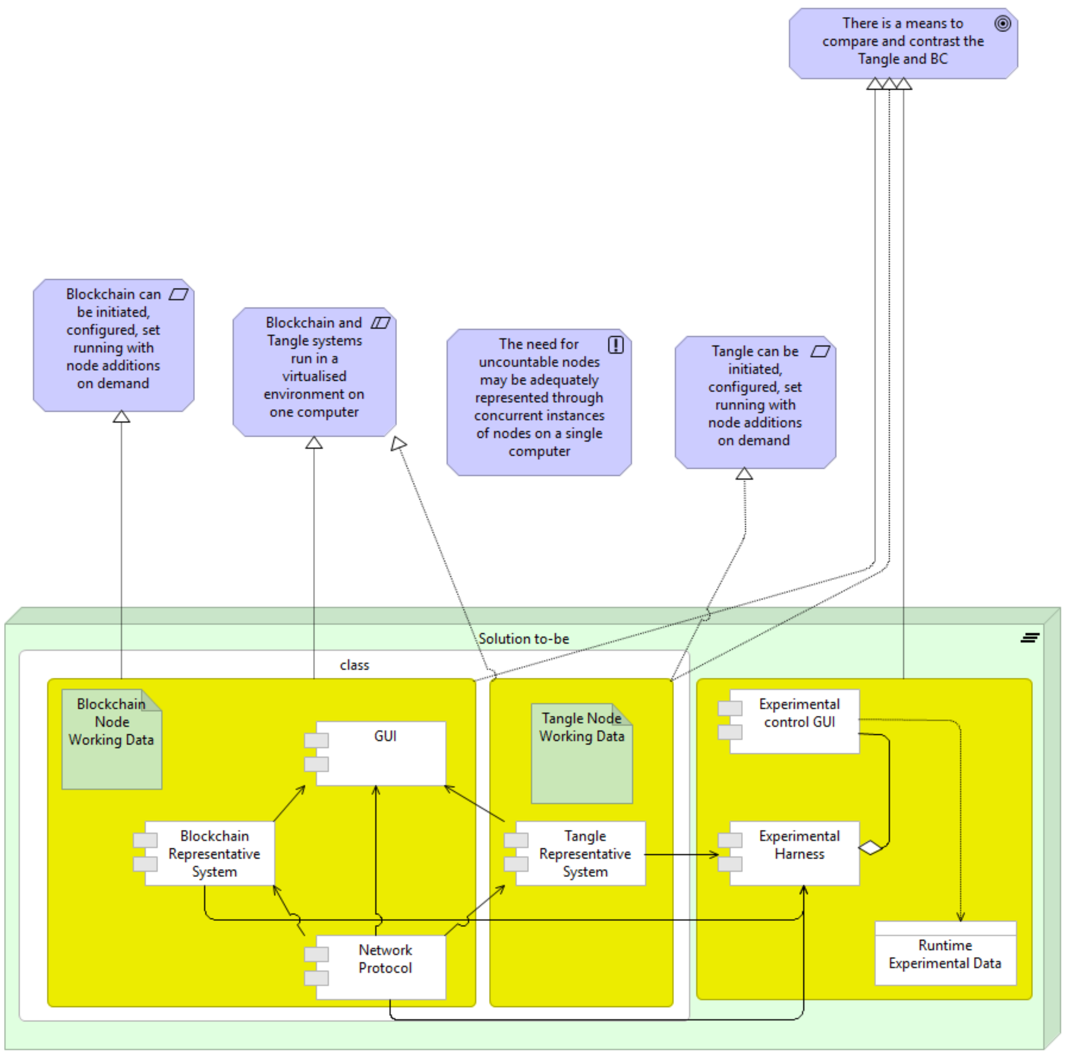
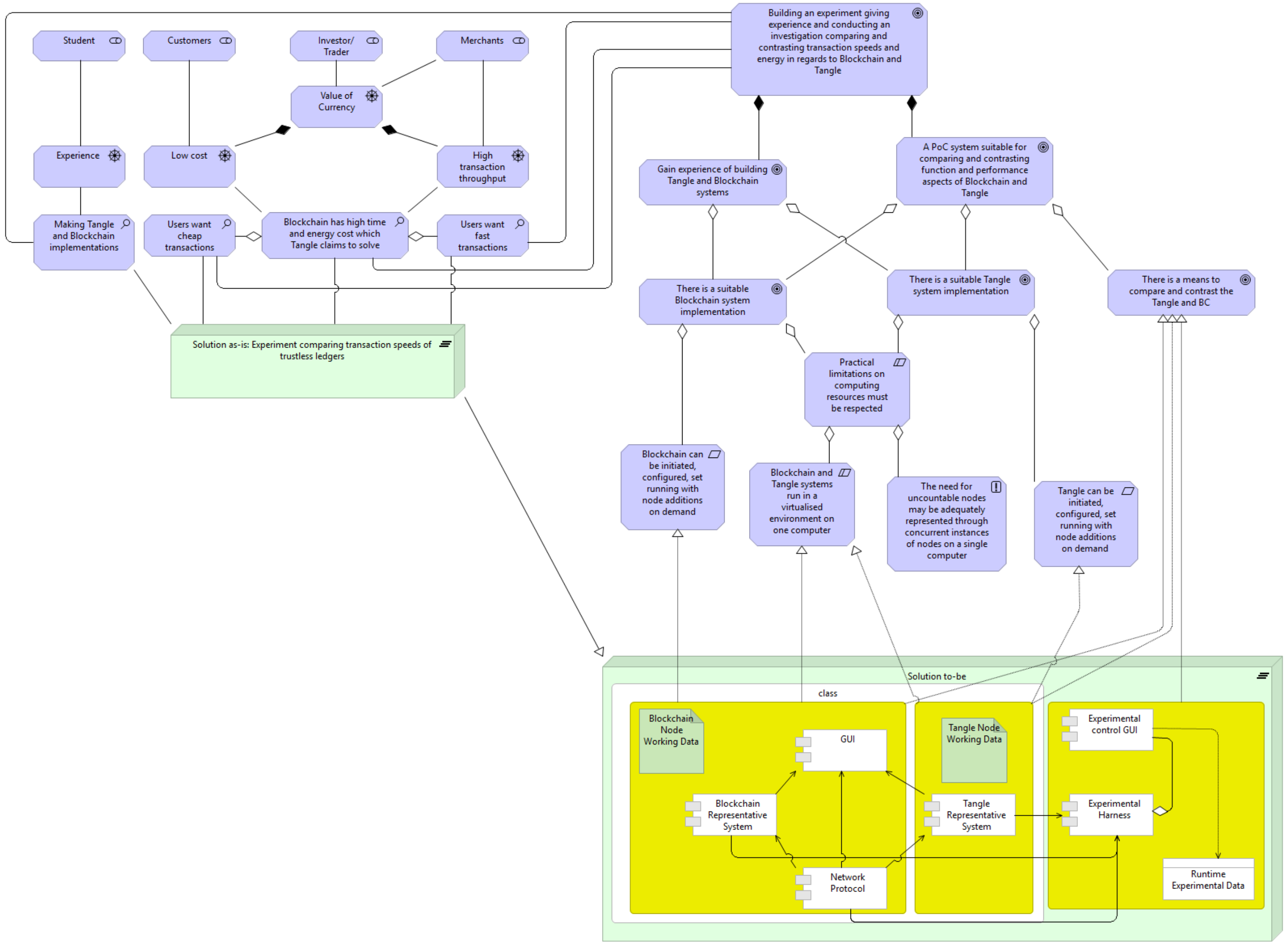

There is a means to compare and contrast the Tangle and BC
Blockchain can be initiated, configured, set running with node additions on demand
Blockchain and Tangle systems run in a virtualised environment on one computer
The need for uncountable nodes may be adequately represented through concurrent instances of nodes on a single computer
Tangle can be initiated, configured, set running with node additions on demand
Solution to-be
class
Blockchain Node Working Data
GUI
Blockchain Representative System
Network Protocol
Tangle Node Working Data
Tangle Representative System
Experimental control GUI
Experimental Harness
Runtime Experimental Data

Figure 14 - *Basis of detailed requirements*

The combined model shows how the solution as-is has transformed into the solution to-be.



Student
Customers
Investor/ Trader
Merchants
Value of Currency
Experience
Low cost
High transaction throughput
Making Tangle and Blockchain implementations
Users want cheap transactions
Blockchain has high time and energy cost which Tangle claims to solve
Users want fast transactions
Solution as-is: Experiment comparing transaction speeds of trustless ledgers
Building an experiment giving experience and conducting an investigation comparing and contrasting transaction speeds and energy in regards to Blockchain and Tangle
Gain experience of building Tangle and Blockchain systems
A PoC system suitable for comparing and contrasting function and performance aspects of Blockchain and Tangle
There is a suitable Blockchain system implementation
There is a suitable Tangle system implementation
There is a means to compare and contrast the Tangle and BC
Practical limitations on computing resources must be respected
Blockchain can be initiated, configured, set running with node additions on demand
Blockchain and Tangle systems run in a virtualised environment on one computer
The need for uncountable nodes may be adequately represented through concurrent instances of nodes on a single computer
Tangle can be initiated, configured, set running with node additions on demand
Solution to-be
class
Blockchain Node Working Data
GUI
Blockchain Representative System
Network Protocol
Tangle Node Working Data
Tangle Representative System
Experimental control GUI
Experimental Harness
Runtime Experimental Data

Figure 15 - *Combined model of preceding analysis and Goal Sketching*

This goal analysis model provides the basis for a solution, which when developed should satisfy the assumptions, constraints and requirements derived in the analysis. Based off these, a model of the solution to-be can be designed in accordance to them. As these objectives are derived from the original goal, the solution will also satisfy this goal. If the final developed system accurately represents this model, then it should be able to fulfil any of the requirements by design. This model is expanded upon below.

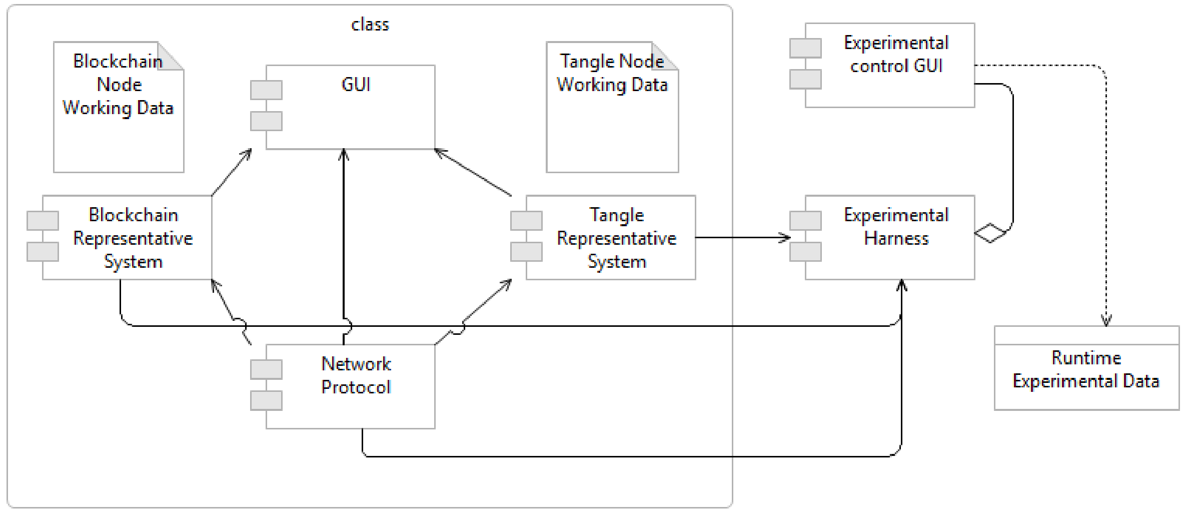

class
Blockchain Node Working Data
GUI
Tangle Node Working Data
Experimental control GUI
Blockchain Representative System
Tangle Representative System
Experimental Harness
Network Protocol
Runtime Experimental Data

Figure 16 - *Larger view of UML class model of both solutions derived using Goal Sketching*

The UML class model presents an implementable solution that can be verified to be correct as it was built using the requirements, constraints, assumptions and goals of this project. The solution is split into two main parts: a class containing the Blockchain and Tangle representative systems which include the components they use to function, and an experiment harness. The class should either have a Blockchain or Tangle representative system within it at one time, as they both require independent implementations from each other.

The GUI is used to control each representative system, forwarding user input and outputting feedback from the system to the user. When a process is made by the representative system which is deemed worthy of sharing, it will be sent to the network protocol to share with other nodes on the network. This protocol will both be able to send messages and receive them while installed in both representative systems. Each class will own its working data; this data will include a full copy of either a Blockchain or a Tangle ledger depending on implementation.

One class represents a single node and the experimental harness should control multiple nodes at a time. The experimental GUI will provide controls to manipulate each node and configure them. It is the harness's job to configure each node after the GUI tells it to. The GUI will also be able to extract runtime results from the nodes and visualise the data.

Both implementations will require their own UML models and dataflow diagrams to classify their exact processes.

Wireframes

Both models should share similar GUIs. They need not be a primary focus but should be able to clearly show functionality and provide useful feedback. A common wireframe for both Blockchain and Tangle is practical as it expected they will have very similar usage but offer different feedback. The network protocol and experimental harness will have their own tabs in addition to the main GUI.

Both models should share a main GUI window that enables a user to:

- Generate a wallet with a public and private key
- see a feedback screen where all functionality related to main GUI returns feedback
- Send ***x*** currency to another public key
- use a button that shows the current balance for a public key
- use buttons that can be used to view the ledger or view certain parts of it
- use a validation button that ensures the ledger is cryptographically valid and transactions are valid

The network protocol should at minimum:

- Show the port the node is currently listening on
- Offer functionality to send a message or ping to other nodes
- Report all network messages sent by the node, and all messages received from other nodes

The experimental harness should at a minimum:

- Allow a user to configure the amount of nodes
- Allow a user to configure the number of transactions generated per minute
- Report node activity (e.g. new transaction generated)

- Report analytics of node activity (e.g. Time between blocks being mined/ transactions added)
- Have buttons to run, stop and create nodes
- Have a button to prompt the harness to analyse data visually using graphs

Main GUI with Network tab open

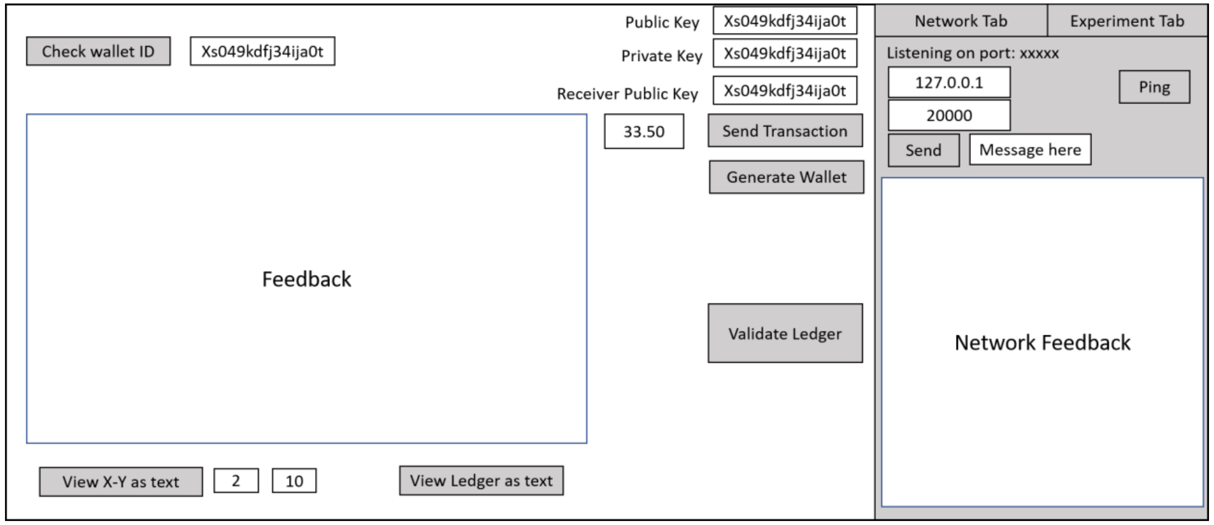


Figure 17 - *Wireframe showing generic main GUI window and network tab*

Main GUI with Experimental tab open

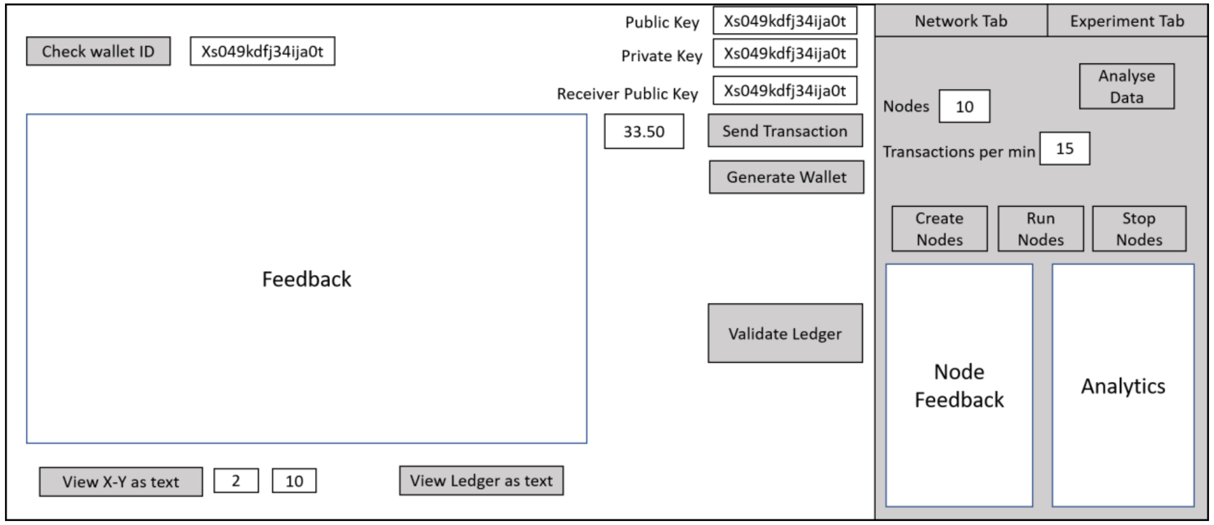


Figure 18 - *Wireframe showing generic main GUI window and experimentation tab*

Both implementations will differ slightly in their actual implementations but following a structure that is as similar as possible can help assure similar functionality and therefore a more trustworthy experiment. C# serves as a useful language for developing relatively simple GUIs such as these. It has a built-in library, Windows Forms, that enables the dragging and dropping of buttons, text boxes and other features easily on a form.

The main screen should allow a user to do the most essential things when running a node or wallet. A typical blockchain wallet UI allows a user to send transactions, check their balance and view their pending/completed transactions. Wallets such as ‘Bitcoin Wallet’ [86] have a similar UI that allows a user to view their transactions, send, receive and check their balance. This wallet has another functionality which views a Bitcoin balance as a fiat balance; but as the currency in this system has no intrinsic value, a similar function would serve no purpose and would be only superficial.

Scaling

A single computer only has limited resources available. An uncountable network of nodes with an immeasurable amount of hashing power cannot be represented without significant scaling. As described above, a typical blockchain network has roughly ~10000 nodes with state-of-the-art hardware; the hashing power is measured in the trillions of hashes per second. A standard PC is only capable of mining at roughly 10-100 thousand hashes per second [87].

It cannot even reproduce the hash rate of a single node. Using the data provided for Blockchain and assuming there are 10000 nodes in a network and 50 trillion hashes per second, then one node produces 5 billion hashes per second. One node has a hash rate ~50000 times faster than a single PC, and the entire network is 500 million times faster.

To compensate for the hash power difference a significant adjustment in block difficulty should be made. Currently in Bitcoin the Proof of Work difficultly requires 18 zeros at the start of the hash. Each zero required increases the complexity of the work by a factor of 16, as it is hexadecimal, each character can be one of 16 values. With a single computer with 100k hashing power/second and 600 seconds of mining (same interval as Bitcoin) it would be reasonable to expect 60 million hashes, which would correlate to a difficultly of 6-7 zeroes. This does not actually need to be considered in the implementation of the Proof of Work as the difficulty algorithm will automatically adjust the difficulty to this scale.

As hashing power does not change the functionality of a Blockchain it appears reasonable to allow a single PC to run a blockchain. This is because the difficulty algorithm will adjust, such that the same number of blocks will always be mined per ***x*** time interval. The ***x*** time interval does not need to be adjusted either; however for the sake of experimentation it would be useful to have a low block time to collect results as fast as possible. A block time of 10-15 seconds will work well as a 10-minute run time should collect roughly 40 blocks.

As the block time has been reduced it would also make sense to reduce the number of transactions stored per block. More transactions per block mean longer read/write operations and memory used in RAM. Minimising the block size will aid performance, but the size needs to be high enough to provide meaningful experimentation. Block size correlates closely with transactions generated per second. IOTA, Bitcoin and Ethereum receive about 5 transactions per second in their networks. The TPS in the network also has a bearing on performance. Therefore to retrieve meaningful experimental results while respecting performance, a TPS of 0.25-0.5 (or 15-30 transactions per minute) and a block size of 15 transactions per block should be used. These numbers are derived by dividing Ethereum's average transactions per block (~150) [88] and its transactions per second (~6.6) [77] by 10 and rounding downwards.

Hashing power is not too relevant for the Tangle, as by design the difficulty will be so low that transactions should be complete almost instantly. 2-3 zeroes in difficulty should be enough to prevent spam and allow fast Proof of Work.

While the system should be optimised for performance, the more nodes running, the more read/write operations and more network activity. Eventually the system will become unstable with too many nodes. A benchmark of 10 nodes running at one time should be set. This benchmark will provide a reasonable network for experimentation and prove the system has adequate performance. Running more than 10 nodes may be possible but could degrade the performance of the GUI noticeably.

Component Design

The solution approach does a component breakdown, where the major components are broken down into their parts. Before implementation, the exact specifications and functionality of these parts should be detailed; in accordance with the literature survey and proposed solution.

The parts of the Blockchain and Tangle representative systems will be specified in their own respective design sections. The common features shared between them are specified below.

<u>Common Features:</u>

Wallet generation

A Blockchain or Tangle wallet typically consists of a pair of public and private keys. These keys should be generated using an industry standard generation algorithm. Otherwise the keys could have integral security issues, which undermines the inherent security of trustless ledgers. RSA is a popular algorithm used for generating key pairs, though ECDSA is preferred given it can generate smaller keys with the same security as RSA [89]. Bitcoin and other Blockchain implementations use ECDSA setting a strong precedent for using ECDSA for wallet generation in both systems.

When a key pair is generated, both keys should be displayed to the user. The private key should be stored in memory for as little time as possible. It is dishonest to keep a copy of the private key, so the system should be designed to terminate the memory space the keys are stored in as soon as they are generated (after displaying them to the user).

Transaction signing and signature verification protocol

Transactions on both Blockchain and Tangle need to be verified by the sender before they can be considered valid. A mechanism needs to be designed to sign the hashes of transactions to create a signature. This signature should be a fixed length string that is verifiable using the public key of the sender. The verification of the signature will also be required as an additional mechanism.

<u>Network Protocol:</u>

Each node needs to able to communicate with one another, to achieve this a network protocol needs to be designed and implemented. Setting up a full network protocol which can talk to different machines is out of the scope of this project. A local network can be realistically set up instead and will still offer the same experimentational utility. The main difference between a local and internet network will be the latency between sending and receiving messages, which is not significant in regard to collecting experimental data.

Each node running will have its own version of the trustless ledger, meaning that all new blocks/transactions must be communicated such that they can update the ledger. These ledgers will be stored separately in folders, which only the respective node can access.

TCP Listener thread

To listen out for any messages that could be sent to the node, a listener is required. To do this, a message listening protocol will need to be chosen. UDP and TCP are native protocols to C# making it relatively simple to utilise one of these. TCP is preferable as it sets up a connection with a sender making it easy to send messages back and forth. This is important, as the messages the node sends often require a response from another node. The TCP Listener will need to have its own dedicated thread, so it can react incoming messages in real-time.

Nodes will each be allocated a port to listen on and the listener will be assigned this port. By default, the first node started will be assigned the port of 20000; this port will increment by one for each successive node.

TCP client send thread

To send a message using TCP, a TCP client should be used. This sets up communication path between the two, where messages can be sent back and forth. A dedicated thread will also be required for the TCP client as messages should be sent as soon as they have been generated/requested.

Messages are configured to be a maximum size of 64KB, this should be large enough to send a single message containing full blocks, transaction pools, connection pools, etc…

Connection Pool of IPs/Ports

Nodes need to be aware of the other nodes on the network. It is impractical that every node on the network will be aware of each other. Instead they should be aware of a some but not all nodes. As the first node is always on port 20000, other nodes can query the first node to check for other nodes on the network. The first node should return a fixed size sample of other nodes to connect to, which the requesting node will then attempt. This sample should be a randomly distributed sample from all nodes on the network; meaning a variety of connections can be made. The length of the sample should be up to 5 nodes, considering that the experimental maximum number of nodes is 10.

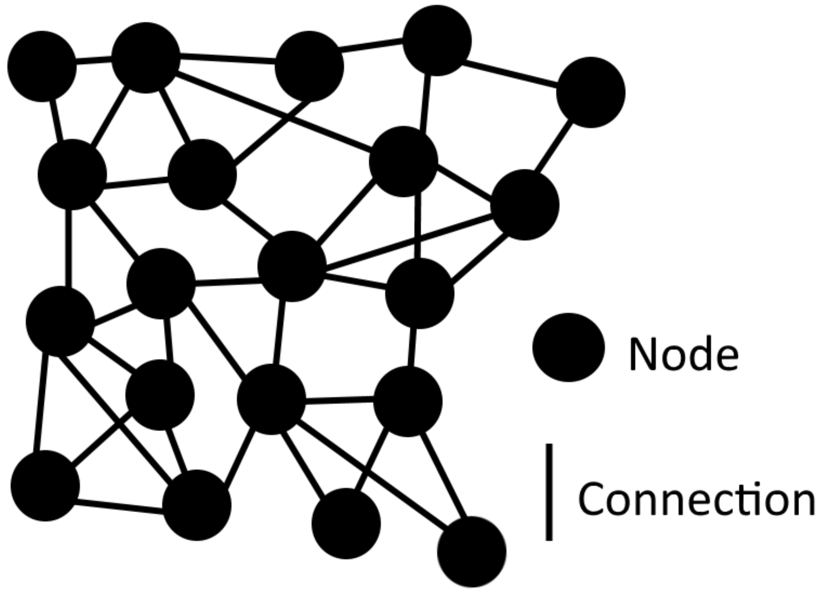


Figure 19 – *Network Structure mock-up, showing a network where nodes are not fully connected.*

After connecting to the network each node should store in memory the nodes it connected to. This area in memory will be the connection pool. Every time an event happens that is worth sharing to the network, such as a new transaction being generated; the event will be shared with the nodes in the connection pool. The receiving nodes should appropriately interpret the event, then propagate the message to nodes in their connection pool if needs be.

It is also the reasonability of all nodes in the network to prevent that the network is not broken. This could occur when a node turns off splitting a connection, leaving two unconnected networks. To prevent this a shutdown mechanism for nodes needs to be implemented. When a node turns off it should warn all other nodes it is about to. Alongside this warning the node should attach its connection pool, which the receiving nodes should utilise. This prevents the network fracturing as the connections of the leaving node are replaced. Once all the receiving nodes have acknowledge the shutdown warning the node can finally turn off.

Message Preparer

Before a message can be sent it needs to be prepared into a send-able and readable format. Messages consist of three sections, an IP (required to send message even in localhost), a port and the message content. The message is then forwarded to a TCP client or listener before it will be sent to the appropriate node.

Message Parser

The message parser is a mechanism that receives an input of a message, interprets the message and performs the necessary functions to fulfil the message. Depending on the type of message the parser also prepares a response.

The full list of expected messages and their content are listed below. Blockchain and Tangle have their own specific messages which will be covered in their respective sections.

Message Formats

REQUEST:

“|REQUEST.NODE.POOL|”

Requests IPs and ports of nodes other connections.

“|REQUEST.NODE.TOTAL|”

Requests number of connections a node has

RESPONSE:

“|RESPONSE.NODE.POOL.{IP,PORT;IP,PORT;IP,PORT;…..;}|”

Responds with all node connected

“|RESPONSE.NODE.TOTAL.amount|”

Responds with number of connections node has.

PING:

“|PING|”

Pings node (starts internal timer)

“|PING.RESPONSE|”

Node response (Ends timer)

SHUTDOWN:

“|SHUTDOWN.NODE.POOL.RESPONSE.{IP,PORT;IP,PORT;IP,PORT;…..;}|”

Warns other nodes it is shutting down, gives all nodes ‘potentially’ new nodes to connect to replace current node with. Nodes reply with SHUTDOWN.ACK

CONNECT:

“|CONNECT|”

Tells receiving node to make connection with sender

“|CONNECT.ACK|”

Acknowledges and approves that connection has been made

“|CONNECT.FULL|”

Replies to connection request informing sending node that receiving node is full and cannot make any more connections.

Error Handling

Listeners and clients are particularly susceptible to exceptions in runtime. Extensive testing and error handling while developing will be required to prevent the network crashing. In particular cases where a message is sent to an inactive port will have to be dealt with. As this will always cause a crash with no error handling.

Experimental Harness:

Configurable Node Controller

The experimental harness should offer reasonable customisation when creating nodes. In its most simple form it should be able to create, run and stop nodes. It should also offer slightly different additional configurations for Blockchain and Tangle.

In the context of Blockchain it should allow customisation of:

- Number of Experimental Nodes
- Transactions generated per minute

- Mining toggle (should the nodes mine)
- Throttle mining toggle (should the nodes mine at a throttled rate)
- Pause after mining block toggle (useful for validation testing)

Additional feedback mechanisms may be required to adequately represent the network is working as intended.

Throttling the amount of hashing power should help normalise the results between different systems with different hashing power.

In the context of the Tangle the experimental harness should offer customisation of:

- Number of Experimental Nodes
- Transactions generated per minute

Mechanisms to control and manipulate multiple node controllers

To control multiple nodes at one time, the implementation of both Blockchain and Tangle models needs to be independent of the UI and controllable by a testing interface. By staying independent multiple objects of Blockchain and Tangle controllers can be initialised at once and controlled by the experiment harness.

This harness should be able to run multiple nodes, configure them and extract important events that occur in each node. When a new transaction is generated, or a new block mined the harness should be notified by the node. It can then collect this information for analysis and also provide feedback to the user.

Analysis mechanisms

The experiment aim of this project is to provide quantitative data regarding the amount of time and energy used per transaction. At a minimum the experiment harness should be able to provide the average energy and time used per transaction. However, energy used is better represented as hashes made per transaction, as this is measurable and comparable.

The analysis of the data should be split into 3 different categories: average time to confirm transaction, average energy (hashes) per transaction and average time since last transaction/block. The latter of these 3 is required to prove that system is working as intended, and generation of transactions and blocks is not artificially skewering the results.

The definition of the average time to confirm a transaction will be customizable by the user. In Blockchain a transaction can be considered confirmed once it is on the chain, or after ***x*** blocks have been added confirming it. Observing the distinction between both definitions, especially the merchant confirm, will show the real extent of time it takes before a transaction is confirmed. In regard to the Tangle, transactions are considered confirmed once they have been confirmed once on the chain; so, time analysis on the Tangle will only require a single definition of confirm.

Visual representations of this data should be made using graphs; where the ***y*** axis is the attribute being measured and the ***x*** axis is the transaction/block count. Alongside this, mean, minimum, maximum and other functions of the data should be provided to give a useful insight.

5.3 Design of Blockchain

The data structure of blocks and transactions should match the diagrams detailed in the literature survey. The data included in them is required at a minimum to create a valid Blockchain system.

Detailed UML Class Diagram

Using the high-level components of the proposed Blockchain implementation, a detailed UML class diagram specifically for Blockchain has been designed.

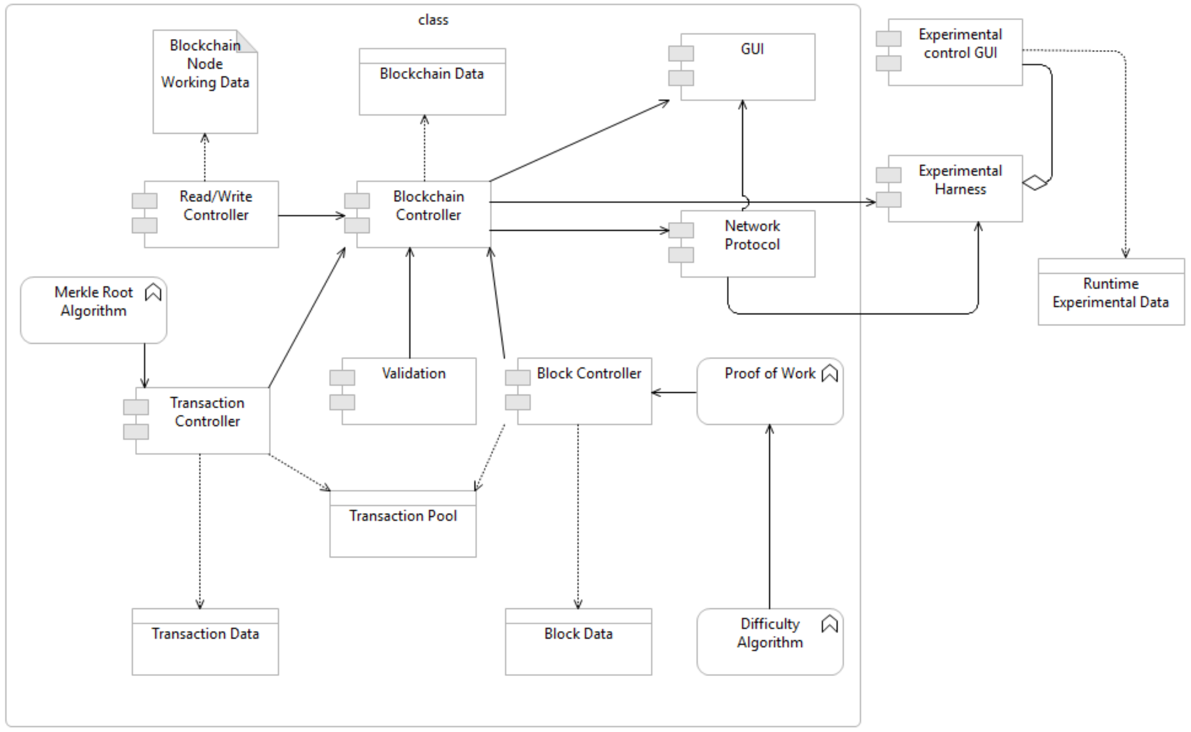


Figure 20 – *Blockchain detailed UML Class diagram*

Blockchain Component Design

This design section should be considered the guidelines for the implementation of the Blockchain. The full implementation is not covered here but will be in the implementation section.

Data Structures

Inside Blockchain there are 3 fundamental data structures that should be utilised. These are the Blockchain itself, a block and a transaction.

The Blockchain is simply a list of every block in the Blockchain. Ordered from the Genesis block to the most recent.

A block contains a list of transactions alongside metadata which can change depending on the Blockchain implementation. As mentioned in the literature survey this data at a minimum should include:

- Index/Height
- Timestamp
- Hash of previous block
- Hash
- Difficulty
- Nonce
- Extra nonce
- Reward
- Merkle Root
- List of transactions (in the same order the merkle root was generated)

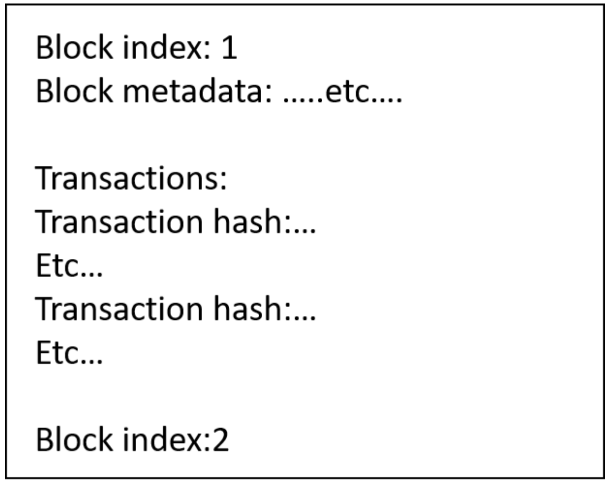
For the purpose of experimentation, no more metadata is needed inside the block, so no more will be added.

A transaction should also contain the same data listed in the literature survey. This data is:

- Hash
- Signature
- To address
- From address
- Amount
- Fee
- Timestamp

Reading and writing mechanism

Blockchain data should be saved to a file/ledger after each update. It should be saved in text and maintain the same order. To achieve this, methods to transform a Blockchain and its parts into text should be implemented. Methods should also be implemented to transform the text back into the parts again. This means all data required in a block/transaction needs to be saved to the ledger, so it can be loaded at a future time.



Block index: 1
Block metadata: …..etc….

Transactions:
Transaction hash:…
Etc…
Transaction hash:…
Etc…

Block index:2

Figure 22 – *Rough guidelines for saving block and transaction information to file*

Ideally loading and saving mechanism should be threaded such that the GUI and mining process is not frozen while waiting for a file to save/load.

Transaction generation

When a user wishes to generate a transaction, they provide their public key, private key, receiver public key, an amount and a fee. This data is required as a prerequisite and is passed to transaction generation method. After validating the public and private key are valid and the sender has enough balance, a timestamp is generated at the time of making the transaction. The timestamp and the other provided data (expect the private key) is inputted into a hash function which returns a hash of the data. This hash is then signed by the private key provided to create a signature. The transaction is then sent to the transaction pool and shared to other nodes on the network.

Block preparation including threaded Proof of work algorithm

Before a node starts mining it must prepare a block. The index of the latest block +1 is assigned to the block and transactions are gathered from the pool. The contents of the block are prepared including: hash of previous block, reward, Merkle Root and difficulty. A threaded Proof of Work of algorithm is then used to find a hash that satisfies the current difficultly.

When mining is finished a timestamp is added at the time of finishing. The block is added to the Blockchain and sent to the connected nodes in the network.

Mining configuration tool

When a node wishes to mine a block, it may wish up to pick transactions from the pool that fulfil certain conditions. The mining configuration tool enables this degree of customisation. The tool should offer the following functionalities.

- Choose maximum amount of transactions to pick up per block
- Choose minimum fee to pick up a transaction
- Choose transactions that are associated with a given public key
- Pick oldest transactions first
- Pick highest fee first
- Pick newest transaction first
- Number of threads used when mining blocks

Transaction Pool

The transaction pool stores pending transactions that have not been added to the Blockchain yet. When a new block is being created it takes transactions from this pool. When the transactions have been added to the Blockchain they are removed from the pool. A transaction pool should have a fixed size. If there are too many transactions in the pool, then the oldest or lowest fee transactions will be removed to make room for new. The transaction pool is stored in the memory of the process meaning when the node is turned off the transactions are deleted. If they are stored in the transaction pool of other nodes, then will not be completely removed from the network until every node has shutdown.

Blockchain specific network messages

UPDATE:

“|UPDATE.NEWBLOCK.BLOCKNUMBER.blockcontents|”

Informs other nodes that a new block has been mined and sends the new block contents. Contents is verified by other nodes. (Other nodes can respond with |UPDATE.NEWBLOCK.ACK|)

“|UPDATE.TRANSACTION.transactioncontents|”

Informs other nodes that new transaction has been added to unconfirmed pool, verified and checked for duplicate. Other nodes can respond with |UPDATE.TRANSACTION.ACK|

REQUEST:

“|REQUEST.BLOCK.TOTAL|”

Requests total number of blocks in blockchain (Used when constructing blockchain)

“|REQUEST.BLOCK.BLOCKNUMBER.x|”

Requests x block from the blockchain.

“|REQUEST.TRANSACTIONPOOL|”

Requests entire transaction pool (Can’t request individual transactions).

RESPONSE:

“|RESPONSE.BLOCK.RESPONSE.x.blockcontents|”

Responds to above request with block x contents which receiving node will verify.

“|RESPONSE.BLOCK.TOTAL.number|”

Responds with number of blocks in the blockchain.

“|RESPONSE.TRANSACTIONPOOL.transactions|”

Contains contents of transaction pool. Format: .{transactioninfo;transactioninfo;….;}

FORK:

If the node has forked from the main chain, then the following commands will be used to correct the chain. The node which has forked will only act upon fork messages when it has recognised it has forked. It will also only accept fork messages from the node it requested the new chain from. Otherwise a malicious node could send forked messages.

“|FORK.REQUEST.blocknumber|”

Requests a block from a node with the given blocknumber (FORK prefix shows intent to fix a fork)

“|FORK.RESPONSE.blocknumber.contents|”

Responds with block requested with FORK prefix so intent of message is clear.

“|FORK.ACK|”

Acknowledgement for FORK messages

Verification Breakdown

To implement a secure Blockchain system the following verification and validation mechanisms should be included.

Transactions on the Blockchain should be validated. This includes:

- verifying the balance of the sender such that it supports their transaction
- verifying the transaction has been signed with sender’s private key
- verifying the hash of the transaction matches the contents
- finally, verifying the transaction is not a double spend by checking if it is already on the Blockchain.

Blocks should be validated too. Their verification methods include:

- verifying the Merkle root of the transaction in a block
- verifying the hash of the block matches the contents
- verifying the previous hash matches the hash of the previous block
- verifying the reward matches the current expected reward
- finally, verifying the mined hash matches the expected difficulty.

The entire chain can be validated by iterating through each block and performing both the transaction and block validations.

Data Flow Diagram

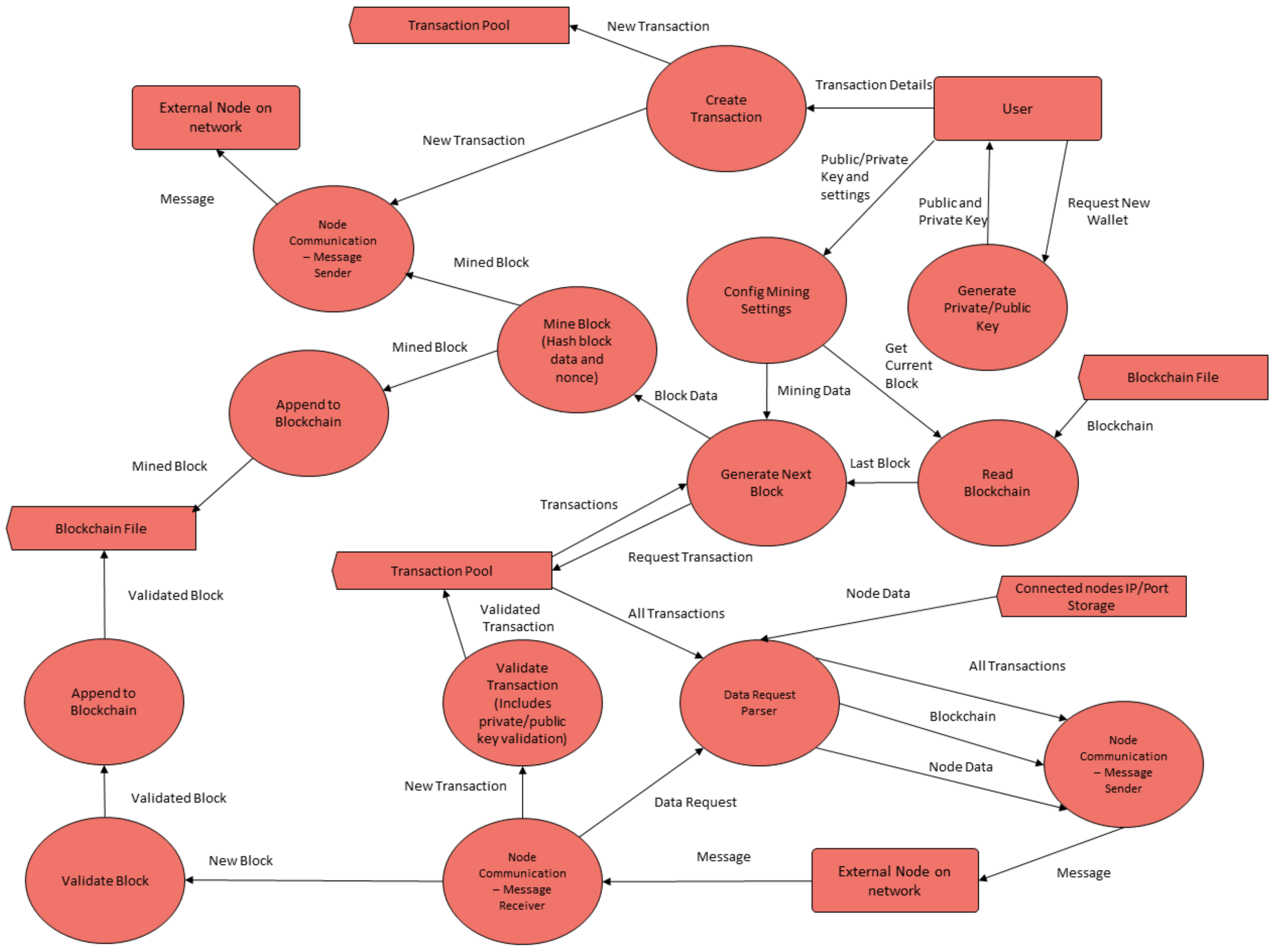


Figure 21 - *Blockchain dataflow diagram without the scope of the experimental harness*

The dataflow diagram demonstrates the complexity of a Blockchain node used in this system. A successful implementation should be able to fulfil every function listed in the diagram.

5.4 Design of the Tangle

The data structure of transactions should match the structure of the diagrams in the literature survey. This is because at a minimum this information is necessary create a valid Tangle transaction and a DAG structure.

Detailed UML Class Diagram

Based off the high-level components of the Tangle implementation a UML Class diagram has been designed.

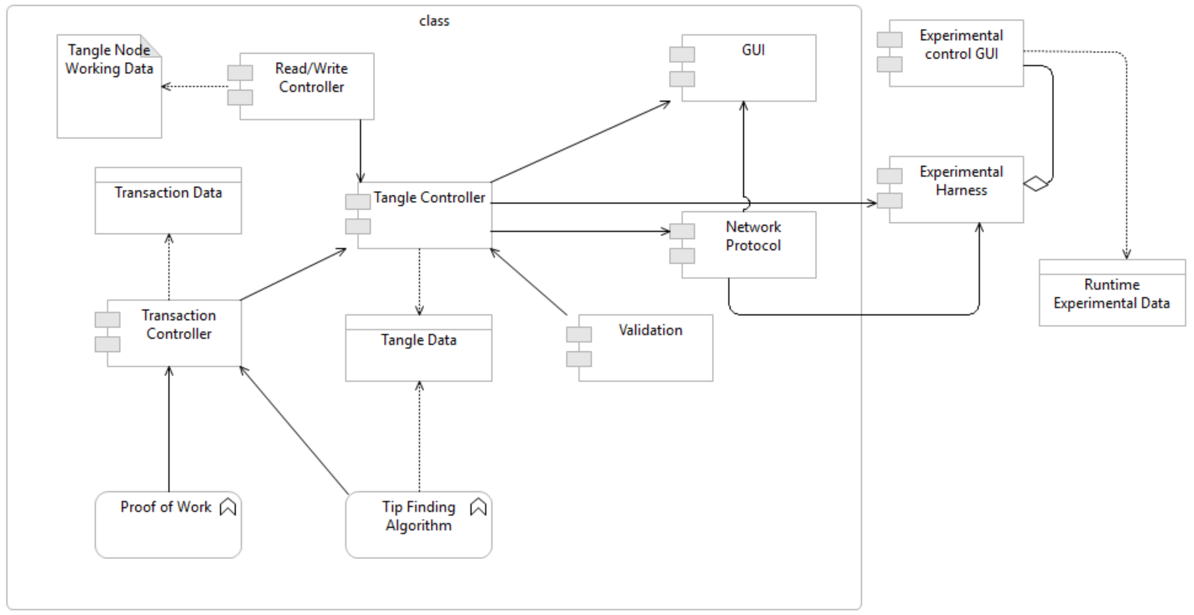


Figure 22 – *Detailed Tangle UML Class diagram*

Tangle Component Design

This design section should be considered the guidelines for the implementation of the Tangle. The full implementation is not covered here but will be in the implementation section.

Data Structures

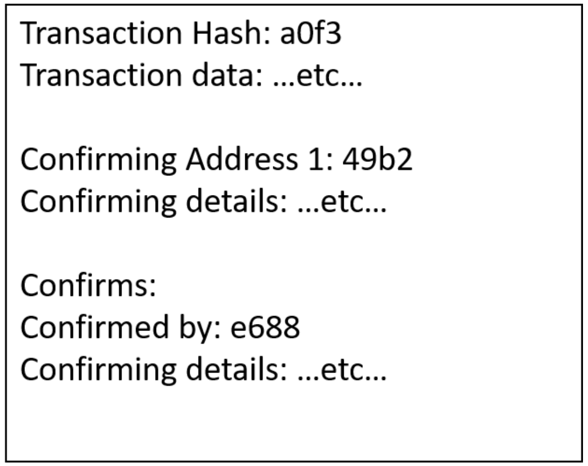
Two fundamental data structures are required in Tangle implementations. First, the Tangle itself, which can be represented computationally as a list of transactions (as the directed connections are inside the transactions). The second data structure being the transactions, which include the following data (as mentioned in the literature survey):

- Hash
- Signature

- Timestamp
- Amount
- To address
- From address
- Height
- Weight
- Confirming (Other transactions that this transaction confirms)
- Confirmed by (Other transactions that confirm this transactions)

Reading and writing ledger from file mechanism

Similarly to Blockchain, the ledger should be written to files and also loaded from files in text. The structure of this data should roughly follow this the format found below:



```
Transaction Hash: a0f3
Transaction data: …etc…

Confirming Address 1: 49b2
Confirming details: …etc…

Confirms:
Confirmed by: e688
Confirming details: …etc…
```

Figure 23 - *Rough guidelines for saving Tangle transaction information to file*

The actual formats for both Tangle and Blockchain will be covered in implementation.

Transaction preparation including double Proof of Work algorithm

A transaction is prepared by generating it using the same procedure mentioned in the Blockchain design section. The user provides their key pairs, sender key and an amount (no fees in tangle). This data is validated, and the transaction is given a timestamp set to the current time. The hash of the transaction is now created by inputting the to address, from address, amount and timestamp into a hash function. This hash is signed using the common signing method to make a signature.

The transaction must now pick two other transactions on the Tangle to confirm. Using a tip selection algorithm two transactions can be found. A height and weight are now assigned to the transaction based off the two transactions picked. A small amount of Proof of Work is done on both transactions (based off the weight attribute) to confirm them. Once the mining for both transactions is finished the transaction is added to the Tangle and shared amongst the network of nodes. Nodes receiving the transaction will validate it and append it to their Tangle.

The transactions that have been confirmed in the Proof of Work should be edited to show they have been confirmed. Future transactions that confirm this newly generated transaction, append their details to the transaction, so the total amount of confirms can be counted easily.

Tangle specific network messages

UPDATE:

“|UPDATE.TRANSACTION.transactioncontents|”

Sends an updates to connected nodes with new transaction and its contents. Responds with UPDATE.ACK

REQUEST:

“|REQUEST.TRANSACTION.hash|”

Requests a transaction present on the Tangle with the given hash.

“|REQUEST.GENESIS|”

Requests the genesis transaction at the start of the Tangle to confirm that both ledgers are built from the same the transactions.

RESPONSE:

“|RESPONSE.TRANSACTION.transactioncontents|”

Sends a transaction and its contents to the requesting node.

“|RESPONSE.GENESIS|”

Sends genesis transaction to the requesting node.

In the case that two nodes have different Tangle ledgers then one will attempt to recursively fix itself it notices it is missing a transaction. For example, if one node reported a new transaction which confirmed a transaction that a different node did not have; the different node would hold onto the new transaction and request the unowned transaction. This process is repeated recursively and then the retrieved transactions are added to the ledger including the new transaction. This process can be achieved purely through using the REQUEST.TRANSACTION and RESPONSE.TRANSACTION messages and additional logic.

Verification Breakdown

To implement a secure Tangle system the following verification and validation mechanisms should be included.

Before a transaction is added to the Tangle by a node it should verify the following properties:

- the sender has the balance to support the transaction
- the transaction has been signed by the sender’s private key
- the transactions hash matches its contents
- the transaction has not been spent already
- the difficulty of the two mined hashes matches the expected difficulty
- the transactions confirmed are real and the hash is valid.

When validating the Tangle each transaction will be verified using the procedure above. Additionally, the DAG structure can be verified by checking that each transaction has confirmed 2 other valid transactions.

Data Flow Diagram

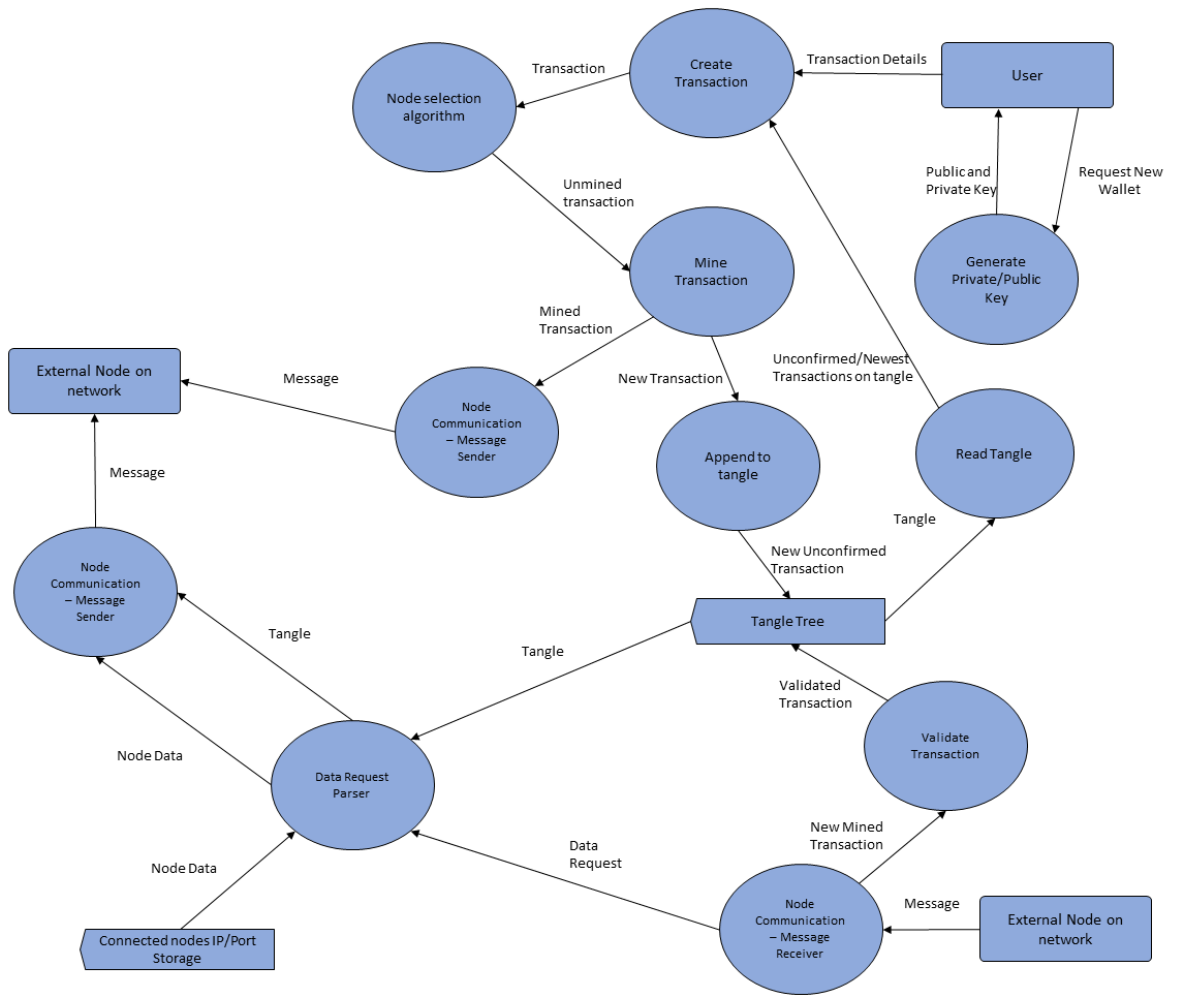


Figure 24 - *Tangle dataflow diagram without the scope of the experimental harness*

The Tangle dataflow diagram is noticeably simpler than the Blockchain dataflow. The ease of Proof of Work and the lack blocks are mainly responsible for this.

6. Implementation

The Implementation stage will detail the process of developing each component in accordance with the design of the system. All components were developed using C# and its native library's. Object orientated design is used to implement the system.

Code snippets will be used throughout to demonstrate how a method has been implemented. There are thousands of lines of code between the two systems, so the entire code cannot be analysed, but the significant parts will.

6.1 Graphical User Interface

Windows forms is native to C# and provides the basic building blocks that can be used to make this experimental tool. The graphical user interface needs to primarily provide an easy user experience. The visual design of the interface is not as important, given that the system is purposed for experimentation. Windows forms elements automatically interfaces with the code; making it easy to directly pipe user inputs and call desired functions.

The UI code has no direct control over the Blockchain/Tangle code but can call methods inside them. Their data is appropriately encapsulated with their classes such that the UI can call public methods but not change any variables which are private to the class. The Windows forms elements used in the GUIs for this system are as follows: Forms, Buttons, RichTextBoxes, TextBoxes, NumericUpDowns, TabControls, CheckBoxes and Charts.

6.1.1 Main UI - Blockchain

The main UI for Blockchain is mainly based off the wireframes from the design. A RichTextBox is used in the left-centre of the screen to show feedback from the Blockchain regarding the users' actions. When a user wishes to view the chain, send a transaction, generate their keys, etc… the feedback text box will report the relevant information back to the user.

This Blockchain implementation has been nicknamed 'TestCoin', a spin on the name Bitcoin deriving from its experimental orientation. Just like Bitcoin, the currency of 'TestCoin' also shares the same name as the system.

The main functions of the Blockchain are mapped to buttons on the main GUI, where you can read from the chain, mine a block, change mining settings, generate a wallet, check the wallet balance, send a transaction, and validate the Blockchain. The following user guide explains the purpose and functionality of each button.

User Guide

Mining Settings – Opens the Mining Settings GUI

Autosave Chain toggle – Toggles the automatic saving to chain when a new block is mined/added

Mine to Wallet – Begins to mine a new Block to the provided address, continues until stopped or completion

Check Wallet ID – Retrieves the available and pending balance of provided public key and the transactions involving the key and reports the information to the feedback textbox.

Send Testcoin – Sends ***x*** TestCoin defined in the 'Amount' input plus a fee ***f*** defined in 'Transaction Fee', to the address inputted in 'Recievers PublicID'. A public key and private key must also be provided to ensure the sender has the balance and to validate the transaction. Depending on the success of the transaction generation, a feedback message is reported.

Validate Private Key – Validates the inputted private key in accordance to the inputted public key. Result output to feedback textbox.

Generate New Wallet – Generates new pair of public and private keys, outputs them in feedback textbox. If Autofill forms is ticked, keys are automatically filled in forms they can be used in.

Get pending transactions – outputs to feedback textbox every transaction in the local transaction pool.

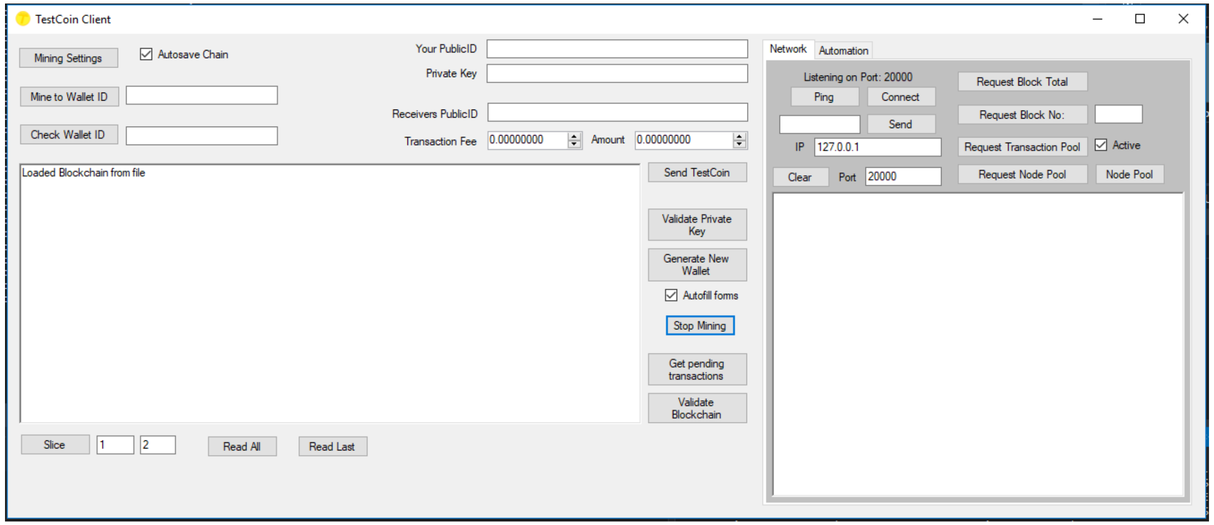
Validate Blockchain – Performs a full validation on the Blockchain and every block within. Outputs result to feedback textbox.

Slice – Takes a slice of the blocks from the chain starting and ending from the given user inputs. Blocks are transformed to text and outputted in feedback textbox.

Read all – Transforms all blocks in chain to text and outputs to feedback textbox.

Read last – Transforms the most recent block on the chain to text and outputs to feedback textbox.

Network and Automation Tabs – Switches between network and automation tabs



☑ Autosave Chain
Transaction Fee 0.00000000 Amount 0.00000000
Loaded Blockchain from file
☑ Autofill forms
Slice 1 2
Listening on Port: 20000
IP 127.0.0.1
Port 20000
☑ Active

Figure 25 – *Main UI for Blockchain based on the wireframes also includes Network Tab*

6.1.2 Main UI – Network Tab

The network tab is located on the right side of the main UI. This tab takes all responsibility for controlling the network and reporting its functionality. When the program is started a port is dynamically assigned to the network. This port is stated in text as 'Listening on Port: xxxxx', starting at port 20000, incrementing up. The feedback involving the network is outputted in the large textbox inside the network tab; this includes any network message sent or received from any node. An IP input is included in the tab despite the network being exclusively local at the moment. This is to enable extension of the network in case of future development.

Many of the controls on the network tab are not necessary to use via client. As they are automated already in the code. They exist mainly for demonstration and tweaking purposes.

User Guide

Ping – Pings the given IP/Port combination with a ping message and starts timer. When a ping response is received the timer is stopped and outputted to the network feedback textbox.

Connect – Attempts to connect to the given node at IP/Port combination. If successful adds node to its connection pool.

Send – Sends the message provided in the input to the given IP/Port combination. Node will attempt to compute message, but if it doesn’t follow network protocol will respond with “|INVALID|”.

Clear – Clears network feedback textbox.

Request Block Total – Requests the total size of the Blockchain from the given IP/Port combination. If the size is larger than this nodes local Blockchain, it will request the missing blocks it does not have and add them to the chain.

Request Block No: - Requests the block and its contents which matches the index given from the given IP/Port combination. If block is not on chain, it is validated then added to chain.

Request Transaction Pool – Requests the transaction pool of the given IP/Port combination. Output each transaction in pool to network feedback textbox. If any of the transactions are missing from this transaction pool they are added (Unlikely as all transactions are automatically broadcasted across network).

Request Connection Pool – Requests the connection pool of the given IP/Port combination. Outputs connections in pool to network feedback textbox.

Node pool – Outputs each connection in this node’s connection pool to network feedback textbox.

Active toggle – Toggles the feedback reporting of sent and received messages. Network messages can be very frequent in busy network, can cause lag to print them out to textbox.

6.1.3 Mining Settings

When a user clicks the Mining Settings button it opens a new UI, which allows them to customise the mining settings. These settings are used applied during the generation of the block to be mined, and the mining itself.

User Guide

Three toggles for transactions to pick up toggles – only one toggle can be selected at a time, they select the type of bias the block generation uses when choosing transactions to include. Highest fee is chosen by default.

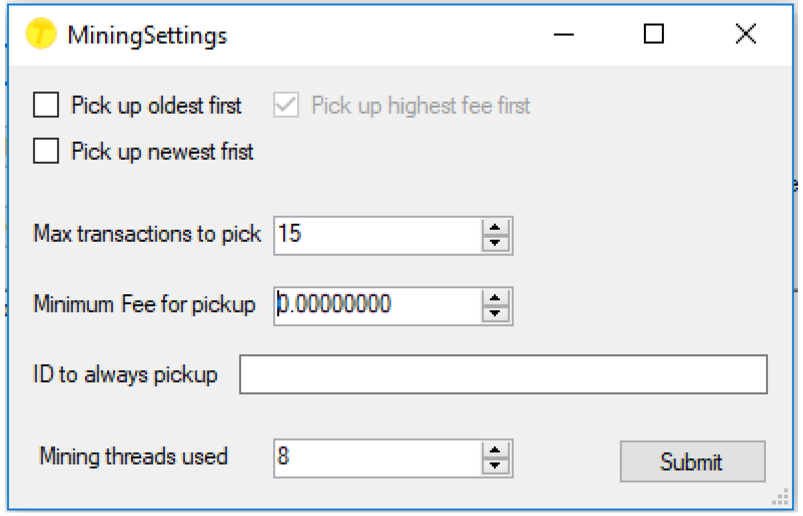
Max transaction to pick – Select maximum amount of transactions that can be included inside a block. 15 by default.

Minimum Fee – The minimum fee of a transaction to pick it up. It altruistic by default and picks up any transaction.

ID to always pickup – Given public key that if included in a transaction, will ensure it gets pick for the block.

Mining threads used – Number of threads used in the mining process. Normally the higher the better, 8 by default.

Submit – Submits these changes and saves them to the MiningSetup object.


MiningSettings

☐ Pick up oldest first
✓ Pick up highest fee first
☐ Pick up newest frist

Max transactions to pick: 15

Minimum Fee for pickup: 0.00000000

ID to always pickup:

Mining threads used: 8

Submit

Figure 26 – *Default MiningSettings UI*

6.1.4 Blockchain Automation Tab

The experimental harness is controlled and configured from the Automation tab. It is host to many controls to design an exact experiment and feedback mechanisms to ensure it is working as intended. The feedback is split across two textboxes. Node info provides the analytics of single nodes and their operations. Analytics provides information about the experimental network of nodes as a whole. For example, a new block is reported in Node info, but the time between new blocks being mined is reported in Analytics.

User Guide

Sync Files – Synchronises the files where ledgers are stored to all match node 20000. Useful for setting up testing on identical ledgers.

Create Nodes – Creates the nodes, allocating their port. Configuration settings for nodes are defined.

Start Nodes – Starts nodes, operation depends off configuration. Will generate transactions and mine blocks if configured.

Stop Nodes – Stop nodes operating.

Delete Nodes – Deletes nodes, stopping them from being ran again.

Reconfig – Nodes are reconfigured based on current configuration settings present in user input.

Read last node: - Reads the last block of node ***x***, providing details about the block in Node Info. If node value is configured to 0, instead outputs the last block index of every node. Useful to check if all blocks are synchronised.

Get Tran: Reports contents of transaction pool of chosen node to Node Info. If detail is toggled, outputs each transaction in pool and first 4 characters of its hash. If detail is not toggled just outputs the number of transactions in the pool.

Analyse Hashes – Opens Analytics UI, which analyses the amount of hashes made per transaction/block

Analyse Block Time – Opens Analytics UI, which analyses the amount of time passing since a block has been mined.

Analyse Confirm Time - Opens Analytics UI, analysing amount of time for transactions to be confirmed. Where the amount of confirms can be configured.

Log toggle – If this toggle is checked, the analysis graph will used logged values, to reduce the effect of outliers on the graph.

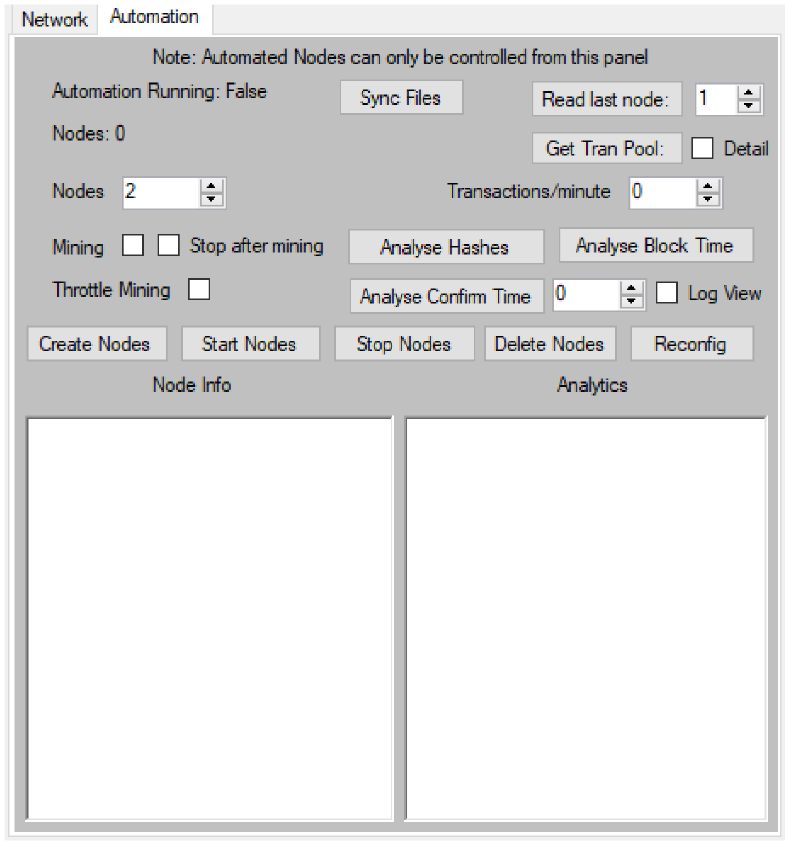
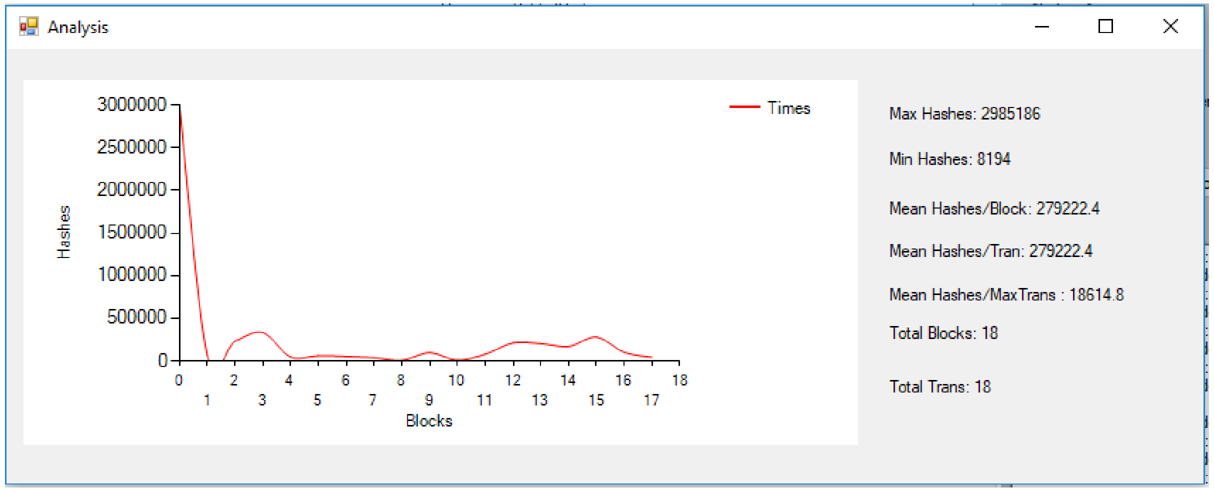


Figure 27 – *Automation tab for Blockchain implementation*

6.1.5 Analysis UI

The Analysis UI comprises of a chart displaying the provided data and a literal analysis of the data. It can handle different data types from both Blockchain and the Tangle and provides the required analysis for each data type. It has little functionality beyond displaying the analytics.



Max Hashes: 2985186

Min Hashes: 8194

Mean Hashes/Block: 279222.4

Mean Hashes/Tran: 279222.4

Mean Hashes/MaxTrans : 18614.8

Total Blocks: 18

Total Trans: 18

Figure 28 – *Analytics UI configured to show block hash data*

6.1.6 Main GUI - Tangle

The main GUI for the Tangle is similar to the Blockchain GUI. It is based of the wireframe and has the same feedback textbox as the Blockchain version. This Tangle implementation has been nicknamed ‘TangleToken’. This name is inspired from two parts; ‘Tangle’ being the type of trustless ledger, and ‘Token’ being IOTAs name for its currency opposed to ‘coin’.

In comparison with TestCoin’s Blockchain GUI, TangleToken is relatively simple. This is due to the absence of blocks and mining, reducing the complexity of adding to the ledger.

User Guide

Check Wallet ID – Retrieves the available and pending balance of provided public key and the transactions involving the key and reports the information to the feedback textbox.

Send TangleToken – Sends ***x*** TangleToken defined in the ‘Amount’ input to the address inputted in ‘Recievers PublicID’. A public key and private key must also be provided to ensure the sender has the balance and to validate the transaction. Depending on the success of the transaction generation, a feedback message is reported. The node will then attempt to add the transaction to the Tangle, by preparing it, doing the proof of work and notifying other nodes.

Validate Private Key – Validates the inputted private key in accordance to the inputted public key. Result output to feedback textbox.

Generate New Wallet – Generates new pair of public and private keys, outputs them in feedback textbox. If Autofill forms is ticked, keys are automatically filled in forms they can be used in.

Validate Tangle – Performs a full validation of the Tangle and all transactions within. Recursively checks transactions confirming/being confirmed. This can take a while on a long chain. The result of this will be outputted to the feedback textbox.

Faucet – As TangleToken is premined it needs to be distributed by the address with all the tokens. The faucet button generates a transaction awarding the given public key 500 tokens. It only does this if the balance of the public key is 0 (To prevent taking advantage of this free currency).

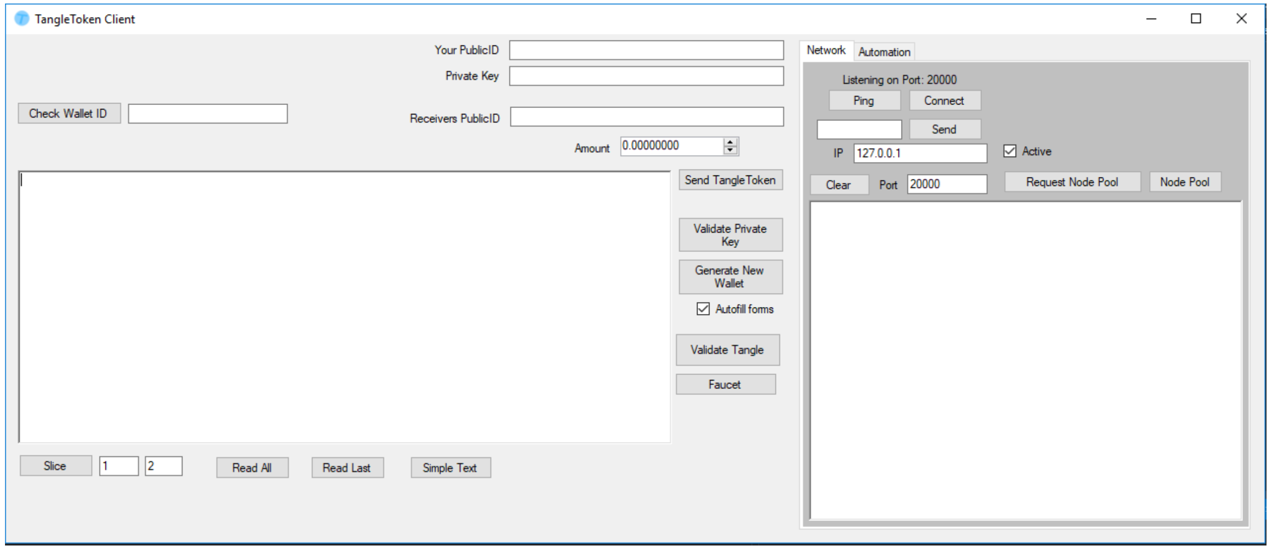
Slice – Selects transactions in ledger from starting slice point to end slice point. Transactions on ledger file are stored linearly in an order therefore this can be done. This order will likely differ from node to node as it is not necessary that ledgers are the same. The transactions will be outputted in text to the feedback textbox.

Read All – Reads every transaction on Tangle ledger and outputs it to feedback textbox.

Read Last – Reads last transaction on Tangle ledger and outputs it to feedback textbox.

Simple Text – Reads every transaction on Tangle but minimises the output to feedback textbox. Output is one line of text per transaction, consisting of: first four letters of hash, transaction height, number of confirms, number of tokens in transaction. This allows easy inspection of the ledger, especially to check if transaction have been confirmed.

The network tab also has some subtle changes too. The Blockchain based buttons such as request transaction pool have been removed.



Amount 0.00000000

Listening on Port: 20000

IP 127.0.0.1 ☑ Active

Port 20000

Slice 1 2

Figure 29 – *Main UI for Tangle based on the wireframes*

6.1.7 Tangle Automation GUI

The Tangle Automation GUI shares many similarities with TestCoin's version. Nodes are configured similarly, and analysis is approached in a similar fashion. However, it is missing multiple feedback mechanisms that TestCoin has. This is mainly due to an increased level of complexity to implement them, therefore were not developed due to time restrictions.

Sync Files – Synchronises the files where ledgers are stored to all match node 20000. Useful for setting up testing on identical ledgers.

Create Nodes – Creates the nodes, allocating their port. Configuration settings for nodes are defined.

Start Nodes – Starts nodes, operation depends off configuration. Will generate and prepare transactions if configured.

Stop Nodes – Stop nodes operating.

Analyse Hashes – Opens Analytics UI, which analyses the amount of hashes made per transaction.

Analyse Timegap – Opens Analytics UI, which analyses the amount of time passing since a transaction has been added to the Tangle.

Analyse Confirm Time - Opens Analytics UI, analysing amount of time for transactions to be confirmed on the Tangle.

Log toggle – If this toggle is checked, the analysis graph will used logged values, to reduce the effect of outliers on the graph.

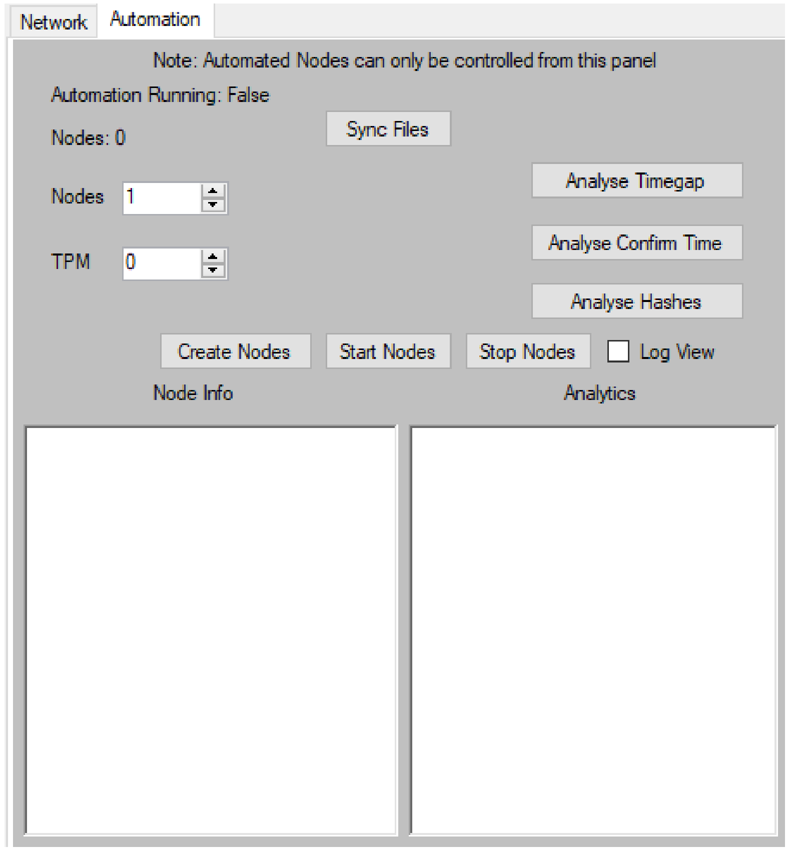


Figure 30 – *Tangle Automation GUI*

6.2 Common Code

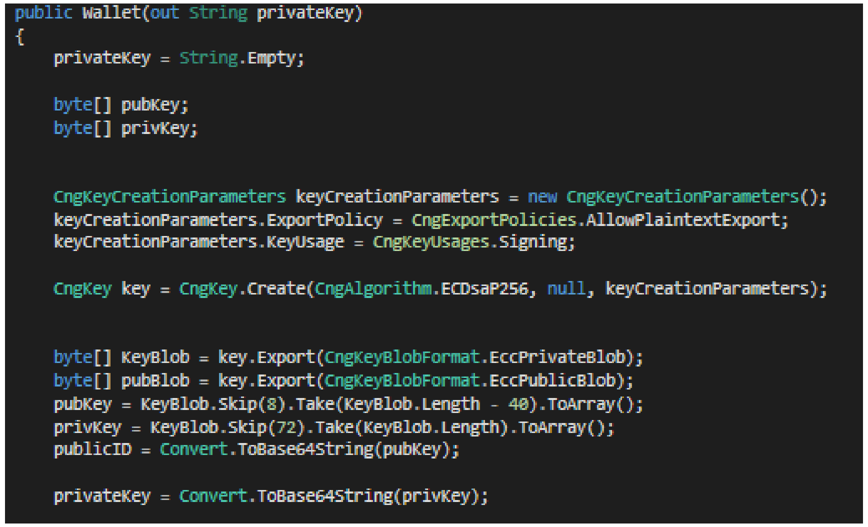
Common code is used between both Tangle and Blockchain implementations. Some of this code is altered slightly for its respective implementation but the majority is the same. This includes the network protocol as the majority of the code is shared.

6.2.1 Wallet Generation and Signing

Public and private key pairs are generated inside the Wallet class. This process begins when a user initialises a wallet object. The code initialises CngKeyCreationParameters which are used to define the usage of the keys being generated, in this scenario asymmetric keys for signing and validation are required. A CngKey is then generated using the parameters and the ECDSA-256 algorithm. The key ‘blob’ is then extracted from the key; this ‘blob’ contains the bytes for the public and private keys.

The key ‘blob’ is 104 bytes bytes long. The first 8 bytes are the header of the data, the public key data starts from byte 8 to 72, making it 64 bytes long. The private key data starts from byte 72 to 104 making it 32 bytes long. This data is copied into public and private byte arrays, where it is then converted into base64 string.

Once the keys are converted to strings, the private key is passed by reference back to the caller. This prevents it from being stored in memory for any longer than necessary. The public key is saved inside the Wallet object and can be accessed publicly.



```
public Wallet(out String privateKey)
{
    privateKey = String.Empty;

    byte[] pubKey;
    byte[] privKey;


    CngKeyCreationParameters keyCreationParameters = new CngKeyCreationParameters();
    keyCreationParameters.ExportPolicy = CngExportPolicies.AllowPlaintextExport;
    keyCreationParameters.KeyUsage = CngKeyUsages.Signing;

    CngKey key = CngKey.Create(CngAlgorithm.ECDsaP256, null, keyCreationParameters);


    byte[] KeyBlob = key.Export(CngKeyBlobFormat.EccPrivateBlob);
    byte[] pubBlob = key.Export(CngKeyBlobFormat.EccPublicBlob);
    pubKey = KeyBlob.Skip(8).Take(KeyBlob.Length - 40).ToArray();
    privKey = KeyBlob.Skip(72).Take(KeyBlob.Length).ToArray();
    publicID = Convert.ToBase64String(pubKey);

    privateKey = Convert.ToBase64String(privKey);
```

Figure 31 – *Wallet object initialisation and key generation*

The Wallet class also includes methods for signing and validating signatures and private keys. Signatures are created by converting a provided hash and key pair to bytes. A built-in method from ECDSA is used to sign the data. This data is converted back to string and returned from the method.

Validation of signatures works almost in the opposite way. A signature and public key are converted to bytes. Then an ECDSA verification method is used to verify that the signature has been signed with the corresponding private key to the provided public key.

6.2.2 Network Protocol

When a node is initialised a ConnectionController is also created. This controller is responsible for handling incoming and outgoing messages, maintaining connections and parsing received messages.

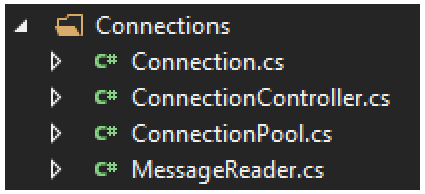
When initialised two threads are started; one to handle incoming messages and the other to send outgoing messages. A TCP Listener is setup to listen for messages indefinitely. Before it can start it needs to find a port. It attempts to listen on port 20000 by default, however if it is taken it will throw an exception. This process is repeated; incrementing the port by one each attempt, until an open port is found. The successful port is assigned to the node. When the TCP listener receives a message, it passes it a MessageReader. The MessageReader parses the message similarly to a compiler and executes the intent of the message. For example, if the message requested the total amount of blocks in the Blockchain; it would call code from the Blockchain to retrieve the answer, then prepare a response message including the answer.

This response is then added to the send thread which is picked up by a TCP client. The message is then sent to the entailed node. The port and IP must be included in the content of the response message, so it can be sent to the correct node.

In the case the TCP listener or TCP client receives or attempts to send a message that is large than the message buffer (64Kb) they will throw exceptions. The code is designed to restart the listeners and clients in case this happens, therefore the network protocol may lose the message but will continue to operate. A single lost message should not harm the system or the nodes.

Connections are maintained in the ConnectionPool. After the ConnectionController is initialised it sends a message to port 20000 (if it is not port 20000) asking for a connection pool. The node at port 20000 will then take a random distributed sample of all nodes. In this current implementation it picks 2 nodes maximum and sends them back to the requesting node. The new node will them attempt to connect to these nodes, adding them to their connection pool. A single node can only have a maximum of 6 connections a one time, so will reject new connection requests pass that point. This ensures that the network is randomly distributed, as not every node can be connected to each other. This does require that the node at port 20000 keeps tabs of online nodes however.

Connections are saved as an object. Their class is Connection, a relatively simple class which stores the IP and port of the connection.



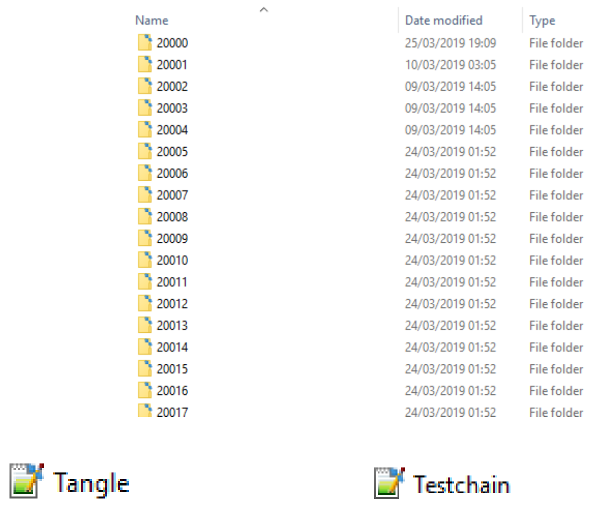
```
◢ Connections
    ▷ C# Connection.cs
    ▷ C# ConnectionController.cs
    ▷ C# ConnectionPool.cs
    ▷ C# MessageReader.cs
```

Figure 32 – *Network protocol classes stored within Connections folder*

The difference between the Tangle and Blockchain implementations is the type of messages expected in the MessageReader. These messages will cause the MessageReader to interact with the Tangle or Blockchain depending on the implementation.

6.2.3 Read/Write mechanisms

Each node on the Blockchain has their own version on the ledger saved on a file. To achieve this each file is stored inside a folder named after their current port. When read/write operations are made the port of the node is used to navigate to the correct file.



Name	Date modified	Type
20000	25/03/2019 19:09	File folder
20001	10/03/2019 03:05	File folder
20002	09/03/2019 14:05	File folder
20003	09/03/2019 14:05	File folder
20004	09/03/2019 14:05	File folder
20005	24/03/2019 01:52	File folder
20006	24/03/2019 01:52	File folder
20007	24/03/2019 01:52	File folder
20008	24/03/2019 01:52	File folder
20009	24/03/2019 01:52	File folder
20010	24/03/2019 01:52	File folder
20011	24/03/2019 01:52	File folder
20012	24/03/2019 01:52	File folder
20013	24/03/2019 01:52	File folder
20014	24/03/2019 01:52	File folder
20015	24/03/2019 01:52	File folder
20016	24/03/2019 01:52	File folder
20017	24/03/2019 01:52	File folder

Figure 33 – *Blockchain and Tangle files and folder structure*

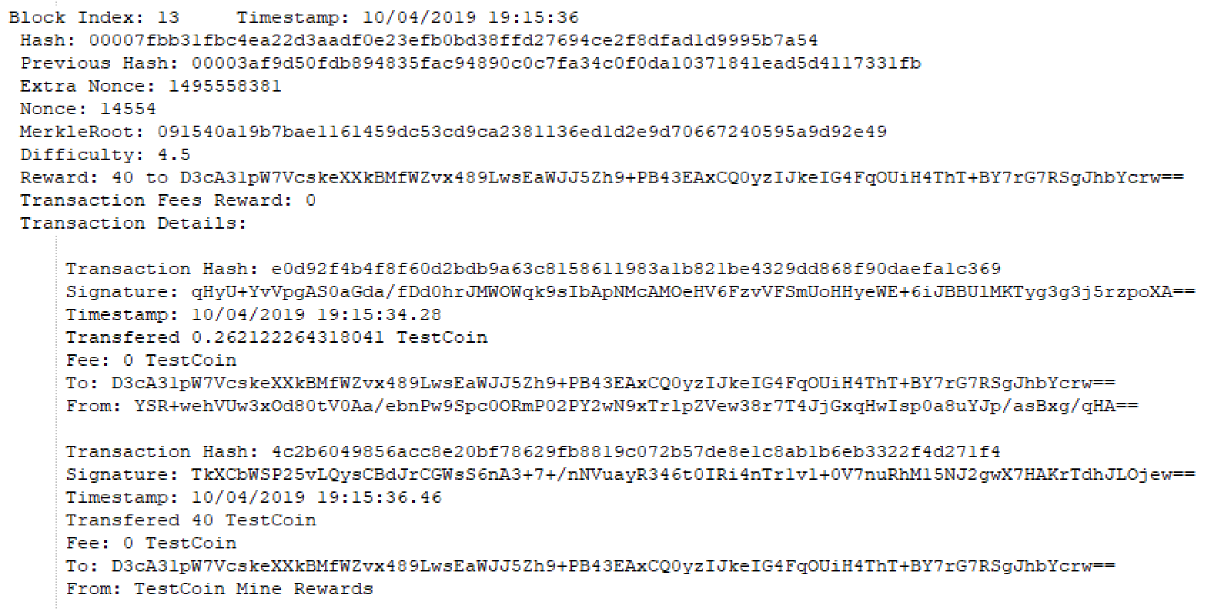
There are 2 primary read mechanisms used across both the Tangle and Blockchain. Both are threaded and return their result to the caller once finished.

Read whole ledger – reads the entire ledger from the file, parsing the text into objects that can be stored in a list of transactions (for Tangle) or list of blocks (for Blockchain).

Read slice – reads entire ledger from file, incrementing a counter each transaction/block. Once the counter is in the slice range, the transactions/blocks are parsed and returned as a list. It is possible to read a single transaction/block using slice.

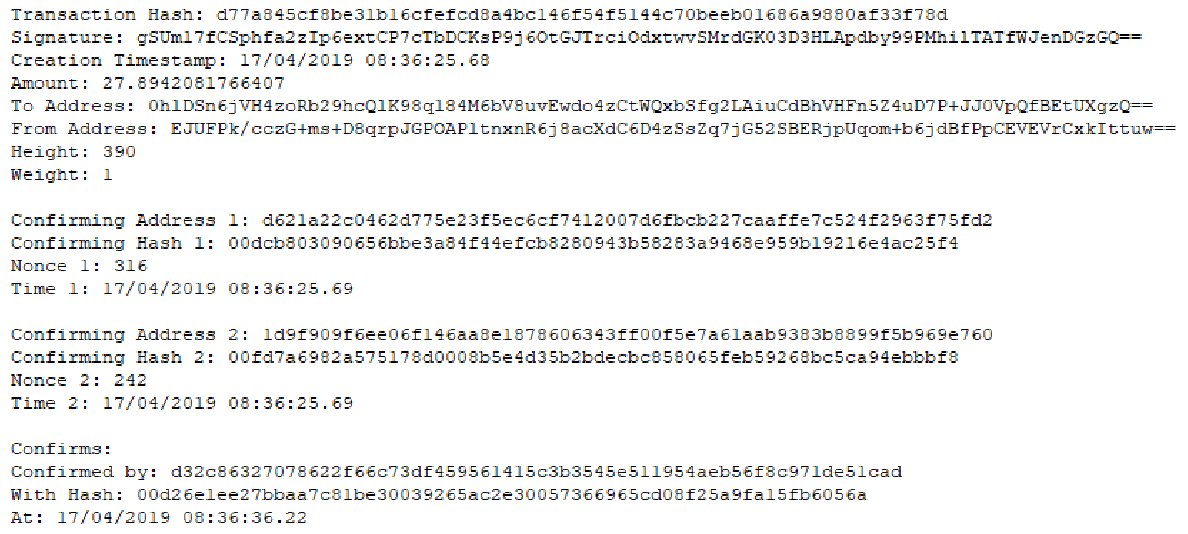
There are 2 types of write mechanisms used; write and append. Write overwrites the entire file and is used for synchronising the ledger files to match node 20000. Append is standardly used to add a new transaction/block to the end of a ledger. It does not overwrite the file, and instead adds to the location in the file specified.

When a block/transaction is written to a file it needs to be transformed into text. There are native methods in transactions and blocks to do this. There variables are slotted into predefined structures, such that they can be easily read back in again.



```
Block Index: 13     Timestamp: 10/04/2019 19:15:36
 Hash: 00007fbb31fbc4ea22d3aadf0e23efb0bd38ffd27694ce2f8dfad1d9995b7a54
 Previous Hash: 00003af9d50fdb894835fac94890c0c7fa34c0f0da10371841ead5d4117331fb
 Extra Nonce: 1495558381
 Nonce: 14554
 MerkleRoot: 091540a19b7bae1161459dc53cd9ca2381136ed1d2e9d70667240595a9d92e49
 Difficulty: 4.5
 Reward: 40 to D3cA31pW7VcskeXXkBMfWZvx489LwsEaWJJ5Zh9+PB43EAxCQ0yzIJkeIG4FqOUiH4ThT+BY7rG7RSgJhbYcrw==
 Transaction Fees Reward: 0
 Transaction Details:

     Transaction Hash: e0d92f4b4f8f60d2bdb9a63c8158611983a1b821be4329dd868f90daefa1c369
     Signature: qHyU+YvVpgAS0aGda/fDd0hrJMWOWqk9sIbApNMcAMOeHV6FzvVFSmUoHHyeWE+6iJBBU1MKTyg3g3j5rzpoXA==
     Timestamp: 10/04/2019 19:15:34.28
     Transfered 0.262122264318041 TestCoin
     Fee: 0 TestCoin
     To: D3cA31pW7VcskeXXkBMfWZvx489LwsEaWJJ5Zh9+PB43EAxCQ0yzIJkeIG4FqOUiH4ThT+BY7rG7RSgJhbYcrw==
     From: YSR+wehVUw3xOd80tV0Aa/ebnPw9Spc0ORmP02PY2wN9xTr1pZVew38r7T4JjGxqHwIsp0a8uYJp/asBxg/qHA==

     Transaction Hash: 4c2b6049856acc8e20bf78629fb8819c072b57de8e1c8ab1b6eb3322f4d271f4
     Signature: TkXCbWSP25vLQysCBdJrCGWsS6nA3+7+/nNVuayR346t0IRi4nTr1v1+0V7nuRhM15NJ2gwX7HAKrTdhJLOjew==
     Timestamp: 10/04/2019 19:15:36.46
     Transfered 40 TestCoin
     Fee: 0 TestCoin
     To: D3cA31pW7VcskeXXkBMfWZvx489LwsEaWJJ5Zh9+PB43EAxCQ0yzIJkeIG4FqOUiH4ThT+BY7rG7RSgJhbYcrw==
     From: TestCoin Mine Rewards
```

Figure 34 – *Blockchain block text structure (with 2 transactions)*



```
Transaction Hash: d77a845cf8be31b16cfefcd8a4bc146f54f5144c70beeb01686a9880af33f78d
Signature: gSUml7fCSphfa2zIp6extCP7cTbDCKsP9j6OtGJTrciOdxtwvSMrdGK03D3HLApdby99PMhilTATfWJenDGzGQ==
Creation Timestamp: 17/04/2019 08:36:25.68
Amount: 27.8942081766407
To Address: 0hlDSn6jVH4zoRb29hcQlK98ql84M6bV8uvEwdo4zCtWQxbSfg2LAiuCdBhVHFn5Z4uD7P+JJ0VpQfBEtUXgzQ==
From Address: EJUFPk/cczG+ms+D8qrpJGPOAPltnxnR6j8acXdC6D4zSsZq7jG52SBERjpUqom+b6jdBfPpCEVEVrCxkIttuw==
Height: 390
Weight: 1

Confirming Address 1: d621a22c0462d775e23f5ec6cf7412007d6fbcb227caaffe7c524f2963f75fd2
Confirming Hash 1: 00dcb803090656bbe3a84f44efcb8280943b58283a9468e959b19216e4ac25f4
Nonce 1: 316
Time 1: 17/04/2019 08:36:25.69

Confirming Address 2: 1d9f909f6ee06f146aa8e1878606343ff00f5e7a61aab9383b8899f5b969e760
Confirming Hash 2: 00fd7a6982a575178d0008b5e4d35b2bdecbc858065feb59268bc5ca94ebbbf8
Nonce 2: 242
Time 2: 17/04/2019 08:36:25.69

Confirms:
Confirmed by: d32c86327078622f66c73df459561415c3b3545e511954aeb56f8c971de51cad
With Hash: 00d26e1ee27bbaa7c81be30039265ac2e30057366965cd08f25a9fa15fb6056a
At: 17/04/2019 08:36:36.22
```

Figure 35 – *Tangle Transaction text structure (with 1 confirm)*

They are also file locking mechanisms in place to stop multiple read/write calls happening at once. When a read/write call is made it joins a queue, when it is at the front of the queue it is executed.

6.3 Blockchain Implementation

6.3.1 Class Structure

The following classes in the Blockchain implementation are the core components that are fundamental to the Blockchain.

Blockchain Class – The Blockchain class is the main controller of the Blockchains functionality. Any operation that is made to the chain is called and controlled from this class. The UI, network protocol and experimental harness all have access to the public methods of this class to manipulate, retrieve and validate data. Additional this class stores the Blockchain as a list of Blocks, another class used in this system. Also stores the transaction pool as a list of transactions.

Block Class – The Block class contains the data of each block, alongside the methods to transforms the data to text, validate the block and complete the proof of work.

Transaction Class – The Transaction class contains transaction data, alongside methods to transform the data to text, and create a hash for the transaction. Verification methods are inside the Block class.

6.3.2 Preparing Blocks

Before a block is mined, the header and transaction data of the block needs to be prepared. This process is started by calling minePendingTrans. The public address of the miner and the mining settings should be provided. The block is prepared by:

- Choose transactions from the transaction pool. The transactions are chosen from the pool in accordance with the mining settings.
- Generating a transaction awarding the reward to the miner address. The reward is calculated based off the total number of blocks mined. In this implementation the initial 20 blocks mined are reward 40 TestCoins, then 30 until block 400, 20 till block 1600000 and 10 for the rest of the blocks. The accumulative fee of the other transactions in this block are also added to this reward.
- The index of the last block is retrieved and its hash, this index +1 is the index of the new block and previous hash of the new block is the hash of the last block.
- Difficulty is calculated using the difficultly algorithm.
- The block is initialised with the transaction data, new index, difficulty, reward and mining settings.

After the block is prepared, the Proof of Work algorithm is started to find the hash which satisfies the difficulty. When the Proof of Work is completed the block is ready to add to the chain. The block is appended to the ledger file and the nodes in the connection pool are notified about the new block. They will add to block to their chains and then notify their connections of a new block. This should cover the entire network rapidly.

In the case a new block is mined by another node while this node is mining; it will stop its mining and add the new block. If two blocks are mined by two nodes at the same time a fork will be created. However, this should be fixed when the next block is mined, as nodes always favour the longest chain.

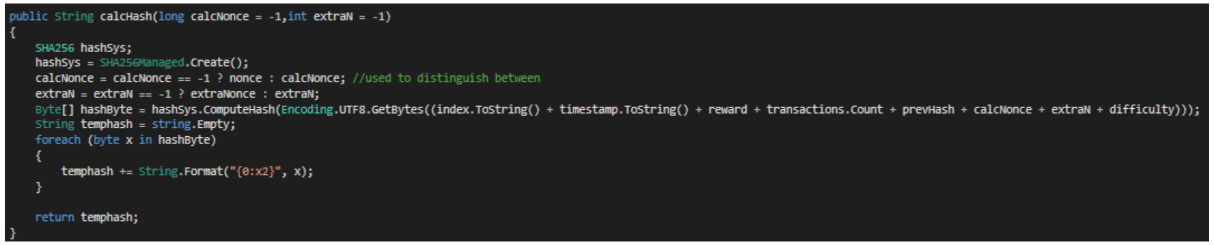
The pending transactions that have been chosen by the block are removed once the block is added to the chain. Nodes that receive the block will also remove the relevant pending transactions.

6.3.3 Proof of Work Algorithm

The Proof of Work algorithm aims to find a hash of the data in a block that satisfies the difficulty provided. Each hash made by the algorithm hashes the following data: block index, timestamp, reward, transaction count, previous hash, nonce, extra nonce and difficulty. The extra nonce is randomly generated based off time and the hash of the miner, meaning work will not be repeated between this node and competing nodes.

In the case the nonce exceeds the max integer value, the nonce is reset to 0 and the extra nonce is incremented. A throttling feature is also added to the algorithm, can which be toggled on and off. For each thread running, after 500 hashes have been made, the thread must sleep for 1.5 seconds. This reduces the total amount of resources used by the PC, meaning tests can be made for extended periods of time without concerns of overheating. This feature is only used by experimental nodes.

Depending on the mining settings up to 8 threads can be ran at once attempting the Proof of Work. Each thread is given a vastly different extra nonce, so they do not repeat work. If any of them succeed, they tell each other to stop and pass the resulting block back to the Blockchain.



```
public String calcHash(long calcNonce = -1,int extraN = -1)
{
    SHA256 hashSys;
    hashSys = SHA256Managed.Create();
    calcNonce = calcNonce == -1 ? nonce : calcNonce; //used to distinguish between
    extraN = extraN == -1 ? extraNonce : extraN;
    Byte[] hashByte = hashSys.ComputeHash(Encoding.UTF8.GetBytes((index.ToString() + timestamp.ToString() + reward + transactions.Count + prevHash + calcNonce + extraN + difficulty)));
    String temphash = string.Empty;
    foreach (byte x in hashByte)
    {
        temphash += String.Format("{0:x2}", x);
    }

    return temphash;
}
```

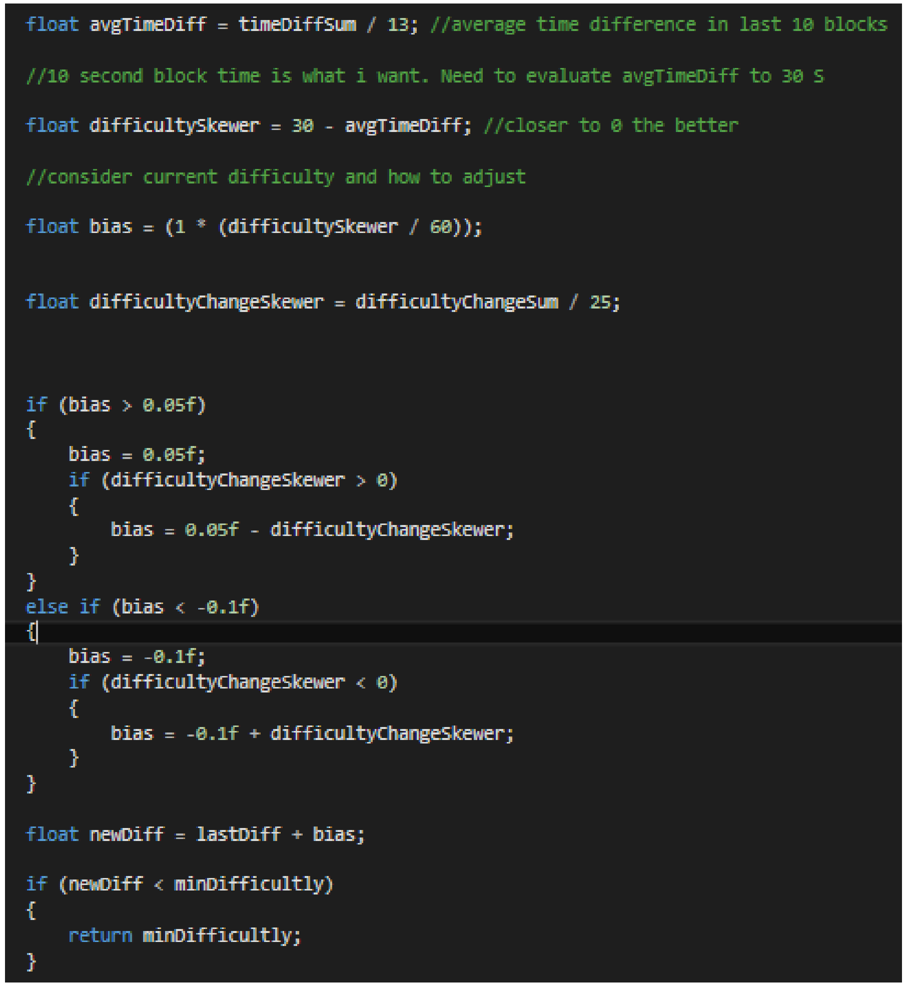
Figure 36 – *calcHash function, hashes block data returning a hash.*

6.3.4 Difficulty Algorithm

Difficulty is calculated based on the time that has passed between the last 10 blocks. The last 10 blocks are retrieved from the ledger and the average time passing between each block is calculated. Ideally, this average time should match the intended time per block. In this implementation that time is 10 seconds. If the average time is more than 10 the difficulty should be reduced, if it is less than 10 the difficulty should be increased. Difficulty is a floating-point number and the change made is based off how close the average time is to the intended time. If it is close, minimal changes will be made. If it is large a larger change is made, however the maximum change that can be made is capped to prevent excessive difficulty changes. If the difficulty is too low or high this change should happen gradually over time, instead of all at once.

The difficulty starts at 4.5 by default for the first 15 blocks. After that the difficulty is calculated using the algorithm. The difficultly cannot get lower than 4 in this implementation, any lower than 4 would be trivial PoW. Infrequent use of the system will result in low difficulty, so this difficulty floor is put in place to prevent that.

The difficulty value correlates to the number of zeros required in the Proof of Work hash. A difficulty of 5 requires 5 zeroes, and a difficulty of 6 requires 6 zeros etc…. The decimal value of the difficulty correlates to the acceptable remaining characters in hexadecimal. For example; with a difficulty of 4.5 a hash would need 4 zeros and a final character of any of the following: 0,1,2,3,4,5,6,7.



```
float avgTimeDiff = timeDiffSum / 13; //average time difference in last 10 blocks

//10 second block time is what i want. Need to evaluate avgTimeDiff to 30 S

float difficultySkewer = 30 - avgTimeDiff; //closer to 0 the better

//consider current difficulty and how to adjust

float bias = (1 * (difficultySkewer / 60));


float difficultyChangeSkewer = difficultyChangeSum / 25;



if (bias > 0.05f)
{
    bias = 0.05f;
    if (difficultyChangeSkewer > 0)
    {
        bias = 0.05f - difficultyChangeSkewer;
    }
}
else if (bias < -0.1f)
{
    bias = -0.1f;
    if (difficultyChangeSkewer < 0)
    {
        bias = -0.1f + difficultyChangeSkewer;
    }
}

float newDiff = lastDiff + bias;

if (newDiff < minDifficultly)
{
    return minDifficultly;
}
```

Figure 37 – *Code sample of the calcDifficulty method; demonstrating how the difficulty is calculated in relation to the average block time.*

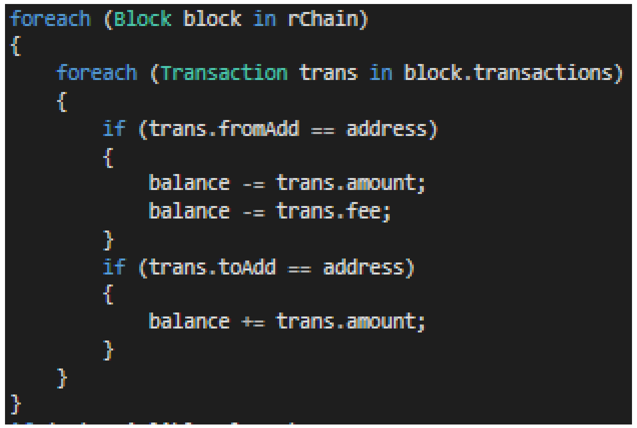
6.3.5 Merkle Root Algorithm

To verify the transactions in the contents of a block are the same the Merkle Root is used. The algorithm iterates through each transaction in a block, pairing them all and then hashing the pairs. The hashes are then iterated through and paired and hashed again. This process is repeated until one hash remains, the merkle root. In the case of an odd hash out, the hash is carried through to the next round of hashing.

6.3.6 Transactions

A user can create a transaction using the UI or an experimental node can generate one. Given that the suitable parameters are provided: valid key pair, amount, fee, receiver address. The key pair is validated using the prior mentioned wallet validation method. To ensure the sender can afford to send the number of TestCoins their balance needs to be checked. A Blockchain method is called to check this.

The getBalance method retrieves the balance for an address. It can be configured to either get an addresses available balance (balance on the chain) or their pending balance (balance on the chain + pending transactions). In this case the available balance is checked to ensure they can afford it. Each block and its transaction are iterated through, searching for a transaction involving the current address. Once a transaction is found their balance is increased or decreased depending on whether they are the sender or receiver of the amount.



```
foreach (Block block in rChain)
{
    foreach (Transaction trans in block.transactions)
    {
        if (trans.fromAdd == address)
        {
            balance -= trans.amount;
            balance -= trans.fee;
        }
        if (trans.toAdd == address)
        {
            balance += trans.amount;
        }
    }
}
```

Figure 38 – *getBalance code fragment which accumulates the balance of an address on the Blockchain*

Once the balance is confirmed as valid the transaction is added to the transaction pool. The transaction is shared with the nodes connections and added to their respective transaction pools. The transaction will stay in the pool until the node is turned off or it is added to a block. A higher fee will normally increase its chances of getting picked.

6.4 Tangle Implementation

6.4.1 Class Structure

The following classes in the Tangle implementation are the core components that are fundamental to the Tangle.

Tangle Class – Includes the all the methods native to retrieving, manipulating and validating the Tangle. Additional stores a 'TangleCache' which stores the ***x*** most recent transactions added to the chain in this session. Storing the entire Tangle in memory should be avoided as it could be very large.

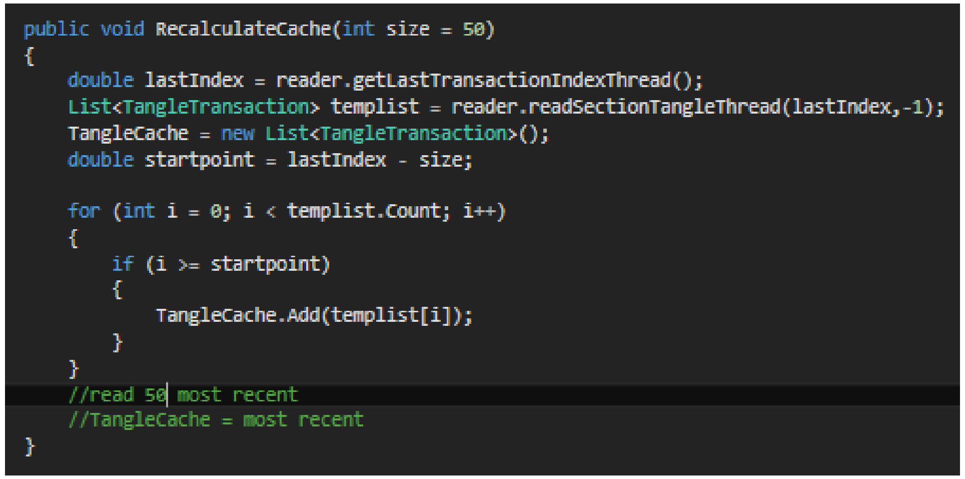
TangleTransaction Class – Transaction class similar to Blockchain Transaction class, however includes additional Tangle data and methods for proof of work, confirming etc…

6.4.2 Directed Acyclic Graph Structure

Storing a directed acyclic graph as a data structure is challenging and unnecessary. Saving this data structure as text into a file would also require an over-engineered implementation. Instead transactions can be stored in a list and the direct connections can be stored inside the structure of the transaction.

When a new transaction confirms two existing transactions, the hash of the existing transactions is stored in the new transactions. As the directed connection is the transaction confirm, the directed acyclic graph structure can be reproduced.

Every Tangle transaction is not stored in memory, instead the ***x*** most recent are stored in a 'TangleCache'. ***x*** is currently defined as 50 by default, with an upper maximum size of 100. When TangleToken is started RecalculateCache is called, the ledger files are read, and the most recent 50 transactions are taken from the ledger and stored in memory. When new transactions are added, they are also added to the 'TangleCache'. This continues until the cache reaches a size over the upper maximum (100). When this happens RecalculateCache is called again, resetting the cache to the most recent 50 transactions again.



```
public void RecalculateCache(int size = 50)
{
    double lastIndex = reader.getLastTransactionIndexThread();
    List<TangleTransaction> templist = reader.readSectionTangleThread(lastIndex,-1);
    TangleCache = new List<TangleTransaction>();
    double startpoint = lastIndex - size;

    for (int i = 0; i < templist.Count; i++)
    {
        if (i >= startpoint)
        {
            TangleCache.Add(templist[i]);
        }
    }
    //read 50 most recent
    //TangleCache = most recent
}
```

Figure 39 – *RecalculateCache method, reads the most recent 50 transactions from the ledger and saves them in memory.*

6.4.3 Transactions

In comparison with Blockchain transactions, Tangle transactions are more complex. They include standard transaction data but also confirmation data. Alongside this there are confirmation and Proof of Work mechanisms in the TangleTransaction class too.

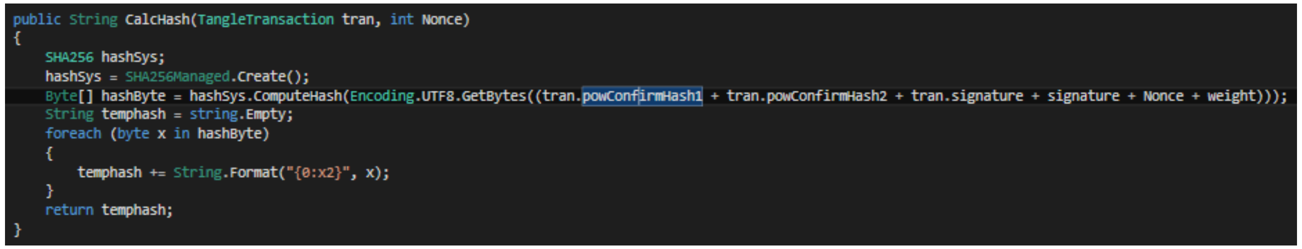
Before a TangleTransaction can be added to the Tangle it needs to be generated and complete its Proof of Work. This process is detailed below:

- The standard data; key pairs, receiver address and amount should be provided. The balance is checked, and private key verified.
- Two tips (transactions) are chosen using the tip finding algorithm.
- A timestamp correlating to the current time is assigned to the transaction.
- The height is dynamically assigned to the transaction; based of the highest height of the two tips +1.
- The weight (Proof of Work difficulty) is statically assigned. In this implementation it is 2 (meaning 2 zeros are required to satisfy a transaction confirmation hash)
- The transaction hash is created using the sender address, receiver address, amount and timestamp.
- The transaction hash is signed using the private key and common signing methods.
- The Proof of Work for the two tips begins. Two threads are created, each assigned one of the two tips. Each attempts to compute the hash to satisfy the difficultly.

- Once the Proof of Work is complete for both tips the transaction is added to the Tangle. Connections are notified of the new transactions, which should propagate across the network.
- The two tips should be edited in the ledger file to add the transaction that confirmed them. (This makes it easy to count the number of confirms a transaction has)

During the Proof of Work for each tip the hash function hashes the following variables:

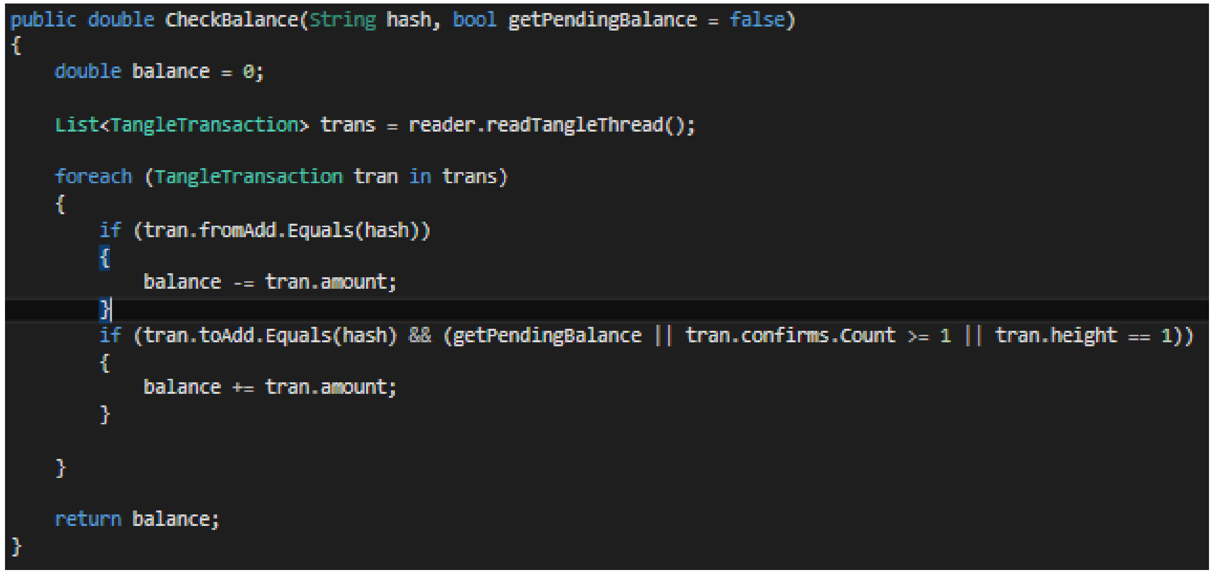
- The proof of work hash of the first transaction the tip confirmed
- The proof of work hash of the second transaction the tip confirmed
- The signature of the tip
- The signature of the transaction doing the work
- An incrementing nonce
- The weight of the transaction



```
public String CalcHash(TangleTransaction tran, int Nonce)
{
    SHA256 hashSys;
    hashSys = SHA256Managed.Create();
    Byte[] hashByte = hashSys.ComputeHash(Encoding.UTF8.GetBytes((tran.powConfirmHash1 + tran.powConfirmHash2 + tran.signature + signature + Nonce + weight)));
    String temphash = string.Empty;
    foreach (byte x in hashByte)
    {
        temphash += String.Format("{0:x2}", x);
    }
    return temphash;
}
```

Figure 40 – *CalcHash; Hash function used in Tangle Proof of Work*

Validating an address has the balance to fulfil a transaction is completed similarly to Blockchain. Every transaction must be temporarily read into memory. The transactions are then iterated through, looking for transactions involving the address as sender or receiver. To be counted transactions require at least one confirmation, unless a flag is otherwise set to true. Transactions that have no confirms are considered ‘pending’ and are not counted towards adding to the sum of the address. However, pending transactions sent by the user will be subtracted from their balance to prevent them sending a transaction that could put them in the red.

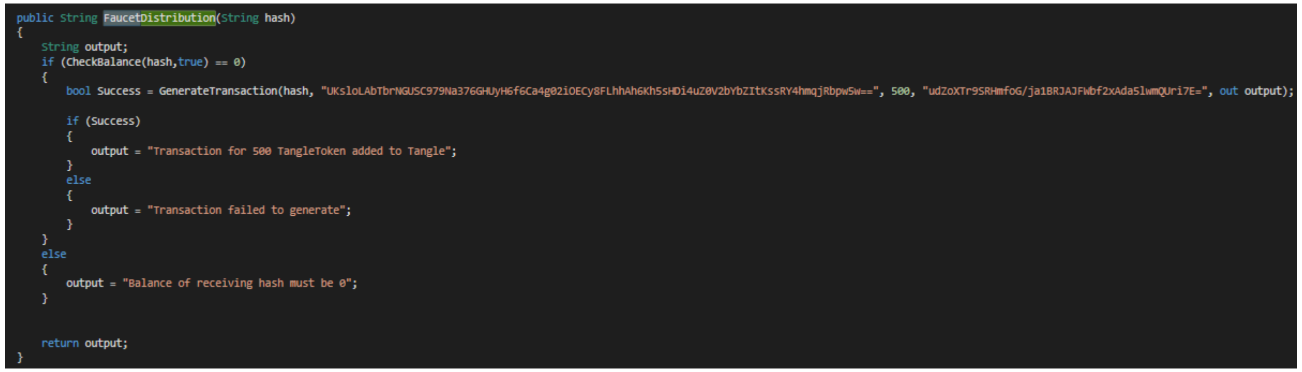


```
public double CheckBalance(String hash, bool getPendingBalance = false)
{
    double balance = 0;

    List<TangleTransaction> trans = reader.readTangleThread();

    foreach (TangleTransaction tran in trans)
    {
        if (tran.fromAdd.Equals(hash))
        {
            balance -= tran.amount;
        }
        if (tran.toAdd.Equals(hash) && (getPendingBalance || tran.confirms.Count >= 1 || tran.height == 1))
        {
            balance += tran.amount;
        }

    }

    return balance;
}
```

Figure 41 – *CheckBalance method; calculates balance of an address by iterating through all transaction on Tangle*

As there is no built-in way of mining or rewarding TangleTokens, they need to be premined. The first transaction on the Tangle should be automatically generated to award an arbitrary address all of the TangleTokens allocated. In TangleToken the total amount of premined tokens is 2 million. This is awarded to a single address. This address has control of every token until they choose to distribute them otherwise. All transactions will be directly connected to this genesis transaction as this is the starting point of the ledger.

To distribute to other addresses a faucet has been implemented. The faucet creates a new transaction granting 500 TangleTokens to a given address. It signs the transaction with the arbitrary public and private key originally defined to receive all tokens. This transaction can only be generated if the balance of given address is zero.



```
public String FaucetDistribution(String hash)
{
    String output;
    if (CheckBalance(hash,true) == 0)
    {
        bool Success = GenerateTransaction(hash, "UKsloLAbTbrNGUSC979Na376GHUyH6f6Ca4g02iOECy8FLhhAh6Kh5sHDi4uZ0V2bYbZItKssRY4hmqjRbpw5w==", 500, "udZoXTr9SRHmfoG/ja1BRJAJFWbf2xAda5lwmQUri7E=", out output);

        if (Success)
        {
            output = "Transaction for 500 TangleToken added to Tangle";
        }
        else
        {
            output = "Transaction failed to generate";
        }
    }
    else
    {
        output = "Balance of receiving hash must be 0";
    }


    return output;
}
```

Figure 42 – *FaucetDistribution; creates new transaction awarding 500 TangleTokens to inputted address (if balance of address is equal to 0)*

6.4.4 Tip finding algorithm

This Tangle uses the methods FindTip and TipSearch to search for tips in the TangleCache. It starts walking from the beginning of the cache exploring one transaction at a time recursively. The depth of each transaction is calculated (the depth is the number of children a transaction is directly and indirectly connected to) which is used in calculating the value of each transaction. When selecting the new transaction to walk to (transactions that have confirmed the current transaction) the depth of the children is considered. A random number is generated, and a transaction is chosen (a higher depth increases the chance of being chosen). This process is repeated until a tip (transaction with no children) is found. The process is repeated twice, ensuring no duplicates.

A transaction added to the end of the DAG has a much higher chance of being picked as the depth of its parents will be higher. This encourages the DAG to be built-up, and in turn increases its cryptographic strength. There is a small chance that a transaction is left unconfirmed and is removed from the TangleCache. If this happens the transaction should be submitted again, as it can never be confirmed, as it cannot be chosen as a tip.

6.4.5 Different Tangles

It is technically possible for two different Tangle ledgers to exist, yet both of them be valid. For example; if one ledger had added a transaction the other ledger had not, they would both still be considered valid. This difference would only be noticed when a new transaction confirms the missing transaction. The node with the missing transaction would query the node which sent the transaction asking for the details of the missing transaction. After receiving it, it would check if there are any missing transactions in this chain. If there is, the process is repeated until otherwise. These missing transactions are stored in a buffer until they have all been received and are then added to the chain. This check is done inside the MessageReader class when a missing transaction is noticed.

Transactions on the Tangle can be in a different order as it makes no difference either way. As long as transactions confirm two other transactions and contain valid header data, they can be in any order they want. Although chronologically order is the most likely and logical.

6.5 Experimental Harness

6.5.1 Controllers and Testers

A single experimental node for TestCoin or TangleToken is controlled with a TangleController or BlockController respectively. These controllers have full access to the nodes and can prompt them to generate transactions, or mine blocks. Call-back methods are used to report back to the controllers, when these events have occurred. The controllers are passed the relevant information about the event.

When a controller is initialised, it initialises its own ConnectionController to join the network, and its own Tangle/Blockchain. A key pair will also be generated for the controller. Therefore, it can make transactions on the ledger. The controller will be provided the configuration set by the user, which controls the transactions generated per minute and if blocks can be mined (and if the mining process is throttled).

A TangleController will be given a TPM (transactions per minute) variable. A thread will be created that will roll a number every second (between 1-60). If this number is lower than the TPM a transaction is generated and added to the Tangle. Once successful, a call-back method will report the transaction to the controller. As the key pairs generated for the controller are new, the address will have 0 balance. The first transaction generated will ask for a faucet transaction. Once this transaction is confirmed and it has its 500 TangleTokens, it will generate random transactions sent to the addreses other experimental nodes on the network.

A BlockController will be given a TPM variable, a mining toggle and a throttle toggle. Like the TangleController, a thread is started which rolls a number to generate transactions. This transaction is passed back to the controller after being generated. The mining toggle decides if a node can mine. If it can a thread will be created, that checks every second if it is mining. If it is not mining, it starts mining a new block. If throttle is toggled, the Proof of Work is throttle. When a block is mined, the key pair generated will be given the reward. When transactions are generated if the key pair has balance higher than 0, if will allocate a portion of the balance to the transaction amount. And has a 50% chance of adding a small fee (1-10% of total amount) too.

BlockTesters and TangleTesters are experimental classes that can initialise multiple BlockControllers and TangleControllers respectively. The user configuration is passed to the tester which then initialises ***x*** number of controllers in accordance with the user's configuration. The controller specific requirements (such as TPM or mining) are passed to the controllers. Testers control multiple controllers, meaning they can collect the experimental data which is passed to the controllers via the Blockchain/Tangle. There are methods inside the testers to create, start and pause all the experiment nodes. BlockTester can also delete and reconfigure the nodes.

6.5.2 Analysis

When the user wishes to analyse the data collected from the experimental nodes they can choose to visualise the data in three formats:

- Hashes per Block/transaction
- Time per Block/transaction
- Time per transaction confirm

This data is all collected inside the tester. Therefore, it can be passed straight to the Analysis class for visualisation on a graph. The Analysis class is different for each implementation, as it expects different data structures from each system. Therefore, they are 6 different analysis methods, which are explained below:

- Hashes per Block/transaction (Blockchain) – Visualises hashes made per block. This data is not read from the block data but is instead passed from the POW algorithm to the controller when mining is finished. Additional analysis includes: Max hashes (of one block), min hashes, mean hashes per block, mean hashes per transaction, mean hashes per transaction (assuming there are 15 transactions in each block), total blocks and total transactions.
- Time per Block (Blockchain) – Visualises time passing since a block has been added to the chain. Additional analysis includes: Max time, min time, mean time, range, total time, total blocks, total transactions.
- Time per transaction confirm (Blockchain) – Visualises amount of time that passed for a transaction to be confirmed ***x*** times. Where ***x*** can be configured by the user. Additional analysis includes: total blocks, confirm value ***x***, max wait, min wait, mean wait.
- Hashes per transaction (Tangle) – Visualises the number of hashes made per transaction. Additional analysis includes: the mean, min and max number of hashes made.
- Time per transaction (Tangle) – Visualises the time gap per transaction added to the ledger.
- Time per transaction confirm (Tangle) – Visualises the amount of time that passed for transactions to be confirmed. Only confirmed transactions are on the graph. Additional analysis includes: total transactions, unconfirmed transactions, mean unconfirmed wait, transactions with 1+ confirms, 2+ confirms and 3+ confirms, mean, min and max wait time for confirm.

6.6 Tests

Validation and verification are intrinsic to a trustless ledger. These methods also double up in utility as they are practical for testing. Almost every component can be verified as functional and working as expected using the validation methods. They can also be somewhat considered automated tests, as they validated things have work as intended. The system will be tested with standard inputs to ensure it works as intended. And the system will be tested with malicious changes/inputs to ensure it catches/deals with them as expected.

Some tests made may not be specific to the Blockchain or Tangle, and therefore should be considered to validate both models in that respect.

6.6.1 Standard Input Testing

Standard procedures and data flows will be followed to ensure the system works as expected.

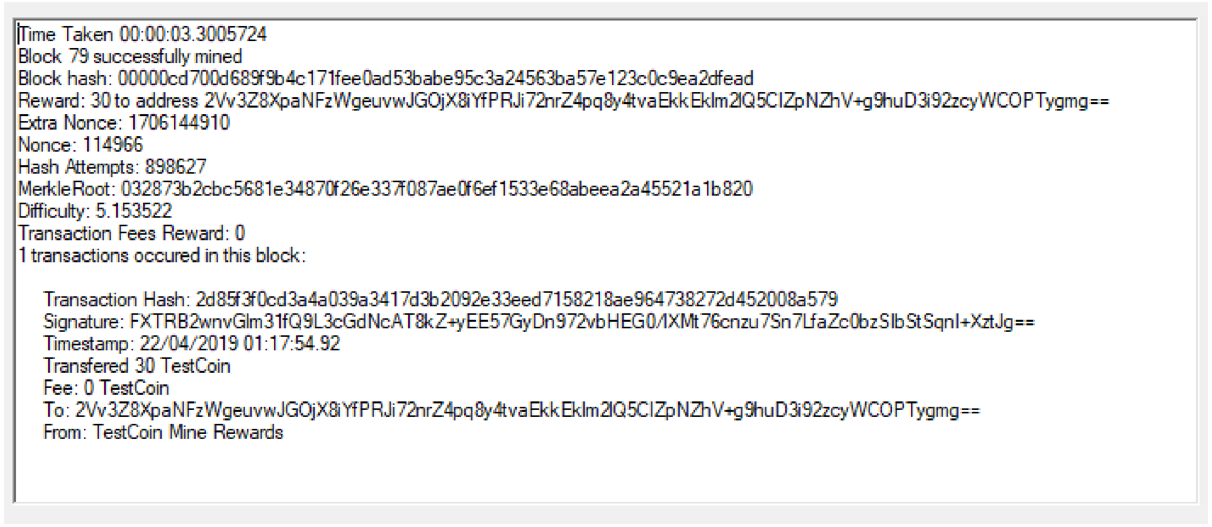
Blockchain

Test: Mine a block to the chain

Steps:

- Generate a new wallet
- Click Mine to Wallet ID
- Wait until mining is complete

Results:



```
Time Taken 00:00:03.3005724
Block 79 successfully mined
Block hash: 00000cd700d689f9b4c171fee0ad53babe95c3a24563ba57e123c0c9ea2dfead
Reward: 30 to address 2Vv3Z8XpaNFzWgeuvwJGOjX8iYfPRJi72nrZ4pq8y4tvaEkkEkIm2lQ5CIZpNZhV+g9huD3i92zcyWCOPTygmg==
Extra Nonce: 1706144910
Nonce: 114966
Hash Attempts: 898627
MerkleRoot: 032873b2cbc5681e34870f26e337f087ae0f6ef1533e68abeea2a45521a1b820
Difficulty: 5.153522
Transaction Fees Reward: 0
1 transactions occured in this block:

    Transaction Hash: 2d85f3f0cd3a4a039a3417d3b2092e33eed7158218ae964738272d452008a579
    Signature: FXTRB2wnvGlm31fQ9L3cGdNcAT8kZ+yEE57GyDn972vbHEG0/IXMt76cnzu7Sn7LfaZc0bzSIbStSqnl+XztJg==
    Timestamp: 22/04/2019 01:17:54.92
    Transfered 30 TestCoin
    Fee: 0 TestCoin
    To: 2Vv3Z8XpaNFzWgeuvwJGOjX8iYfPRJi72nrZ4pq8y4tvaEkkEkIm2lQ5CIZpNZhV+g9huD3i92zcyWCOPTygmg==
    From: TestCoin Mine Rewards
```

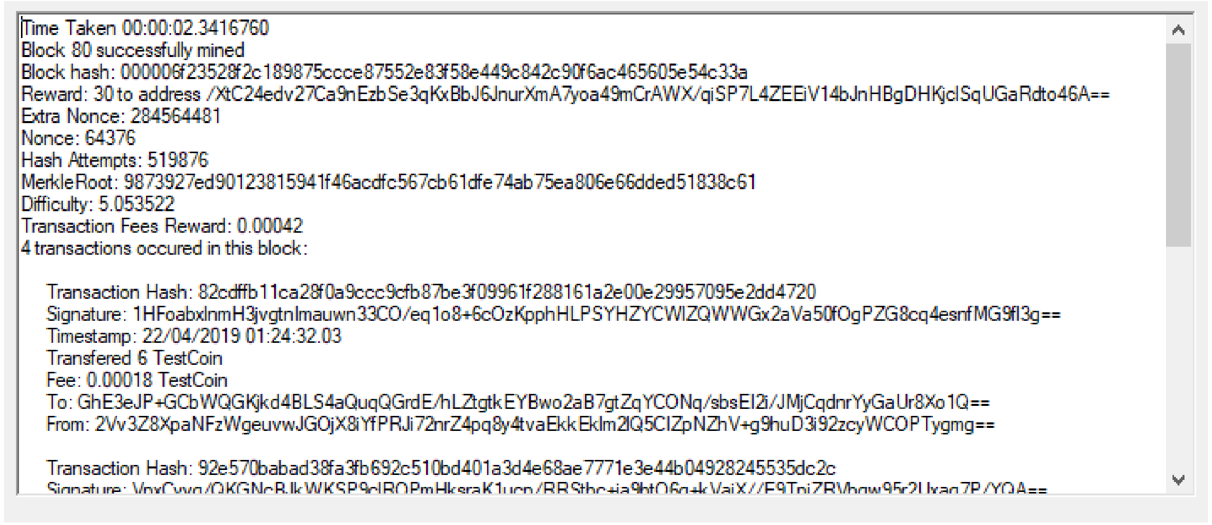
Block successfully mined to chain, address is given appropriate reward.

Test: Generate transactions on one node, and have another node add them to a block on the chain

Steps:

- Open two TestCoin clients
- On one client should generate 3 transactions
- The second client should receive the transaction via the network
- The second client should mine a block, with these transactions included
- Wait until mining complete

Results:



```
Time Taken 00:00:02.3416760
Block 80 successfully mined
Block hash: 000006f23528f2c189875ccce87552e83f58e449c842c90f6ac465605e54c33a
Reward: 30 to address /XtC24edv27Ca9nEzbSe3qKxBbJ6JnurXmA7yoa49mCrAWX/qiSP7L4ZEEiV14bJnHBgDHKjclSqUGaRdto46A==
Extra Nonce: 284564481
Nonce: 64376
Hash Attempts: 519876
MerkleRoot: 9873927ed90123815941f46acdfc567cb61dfe74ab75ea806e66dded51838c61
Difficulty: 5.053522
Transaction Fees Reward: 0.00042
4 transactions occured in this block:

    Transaction Hash: 82cdffb11ca28f0a9ccc9cfb87be3f09961f288161a2e00e29957095e2dd4720
    Signature: 1HFoabxlnmH3jvgtnlmauwn33CO/eq1o8+6cOzKpphHLPSYHZYCWIZQWWGx2aVa50fOgPZG8cq4esnfMG9fl3g==
    Timestamp: 22/04/2019 01:24:32.03
    Transfered 6 TestCoin
    Fee: 0.00018 TestCoin
    To: GhE3eJP+GCbWQGKjkd4BLS4aQuqQGrdE/hLZtgtkEYBwo2aB7gtZqYCONq/sbsEI2i/JMjCqdnrYyGaUr8Xo1Q==
    From: 2Vv3Z8XpaNFzWgeuvwJGOjX8iYfPRJi72nrZ4pq8y4tvaEkkEklm2lQ5CIZpNZhV+g9huD3i92zcyWCOPTygmg==

    Transaction Hash: 92e570babad38fa3fb692c510bd401a3d4e68ae7771e3e44b04928245535dc2c
    Signature: VpxCvwq/QKGNcBJkWKSP9clROPmHksraK1ucn/BRSthc+ia9htO6q+kVaiX//E9TpiZBVbqw95r2Uxaq7P/YQA==
```

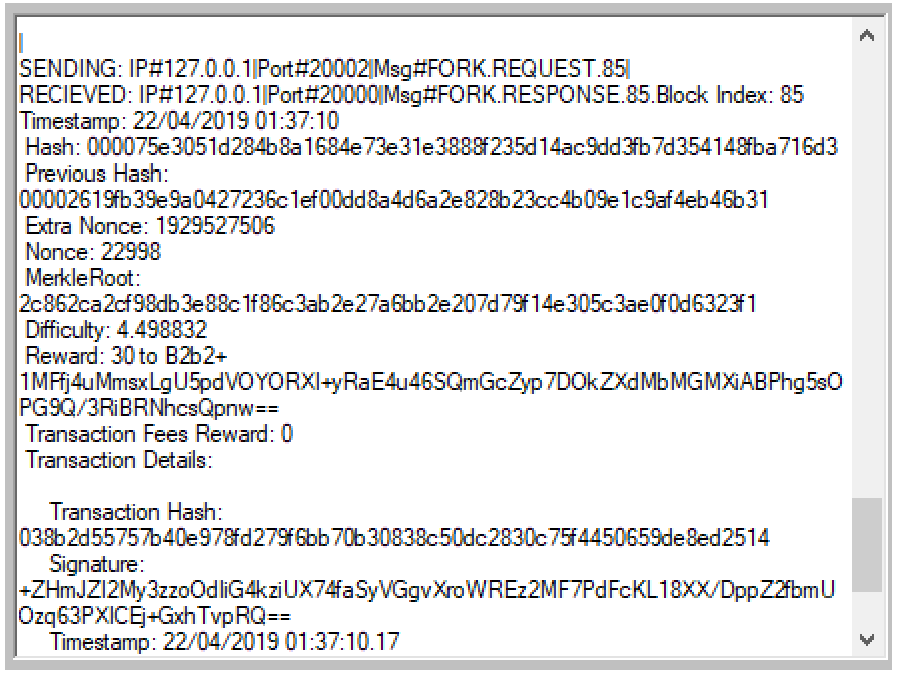
All 3 transactions are included in the block mined on the other client.

Test: Automatic fixing of a fork; between two nodes with different ledgers

Steps:

- Open two clients which have different ledgers to each other
- When a new block is mined they should automatically detect a difference, then one of them should fix to match the other.

Results:



```
|
SENDING: IP#127.0.0.1|Port#20002|Msg#FORK.REQUEST.85|
RECIEVED: IP#127.0.0.1|Port#20000|Msg#FORK.RESPONSE.85.Block Index: 85
Timestamp: 22/04/2019 01:37:10
 Hash: 000075e3051d284b8a1684e73e31e3888f235d14ac9dd3fb7d354148fba716d3
 Previous Hash:
00002619fb39e9a0427236c1ef00dd8a4d6a2e828b23cc4b09e1c9af4eb46b31
 Extra Nonce: 1929527506
 Nonce: 22998
 MerkleRoot:
2c862ca2cf98db3e88c1f86c3ab2e27a6bb2e207d79f14e305c3ae0f0d6323f1
 Difficulty: 4.498832
 Reward: 30 to B2b2+
1MFfj4uMmsxLgU5pdVOYORXI+yRaE4u46SQmGcZyp7DOkZXdMbMGMXiABPhg5sO
PG9Q/3RiBRNhcsQpnw==
 Transaction Fees Reward: 0
 Transaction Details:

    Transaction Hash:
038b2d55757b40e978fd279f6bb70b30838c50dc2830c75f4450659de8ed2514
    Signature:
+ZHmJZI2My3zzoOdliG4kziUX74faSyVGgvXroWREz2MF7PdFcKL18XX/DppZ2fbmU
Ozq63PXICEj+GxhTvpRQ==
    Timestamp: 22/04/2019 01:37:10.17
```

The clients notice and send ‘FORK’ messages between themselves to fix the fork.

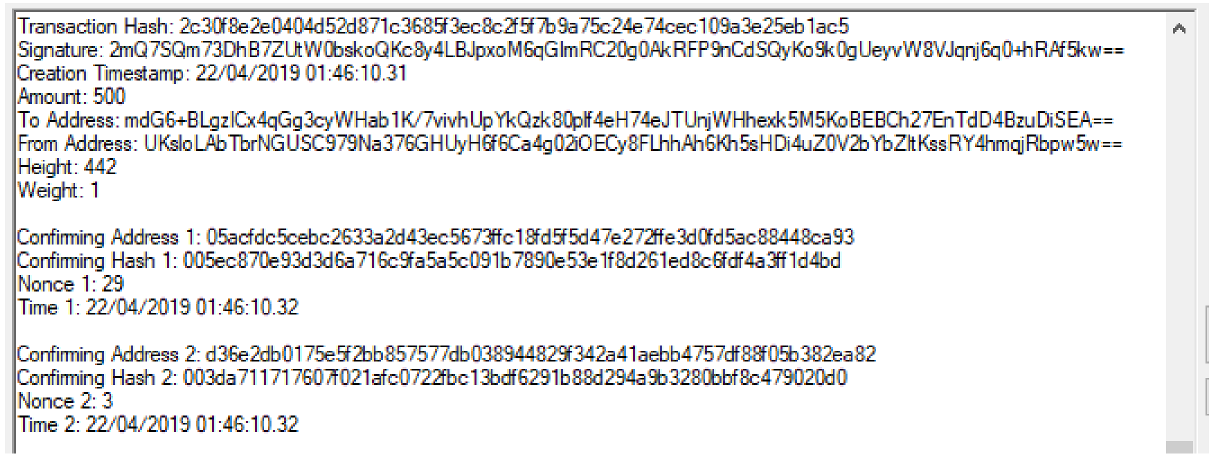
Tangle

Test: Confirm a faucet transaction

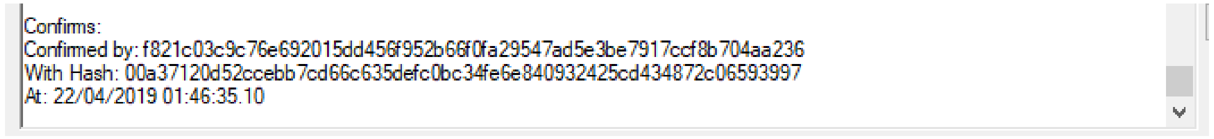
Steps:

- Generate a new wallet
- Generate a new faucet transaction
- Continue generating transactions until the faucet transaction has been confirmed

Results:

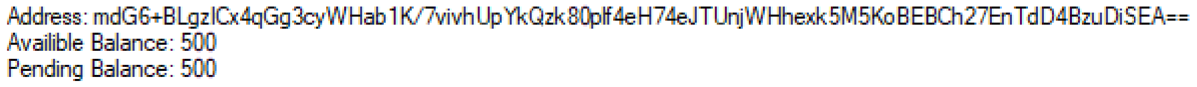


```
Transaction Hash: 2c30f8e2e0404d52d871c3685f3ec8c2f5f7b9a75c24e74cec109a3e25eb1ac5
Signature: 2mQ7SQm73DhB7ZUtW0bskoQKc8y4LBJpxoM6qGlmRC20g0AkRFP9nCdSQyKo9k0gUeyvW8VJqnj6q0+hRAf5kw==
Creation Timestamp: 22/04/2019 01:46:10.31
Amount: 500
To Address: mdG6+BLgzICx4qGg3cyWHab1K/7vivhUpYkQzk80plf4eH74eJTUnjWHhexk5M5KoBEBCh27EnTdD4BzuDiSEA==
From Address: UKsloLAbTbrNGUSC979Na376GHUyH6f6Ca4g02iOECy8FLhhAh6Kh5sHDi4uZ0V2bYbZltKssRY4hmqjRbpw5w==
Height: 442
Weight: 1

Confirming Address 1: 05acfdc5cebc2633a2d43ec5673ffc18fd5f5d47e272ffe3d0fd5ac88448ca93
Confirming Hash 1: 005ec870e93d3d6a716c9fa5a5c091b7890e53e1f8d261ed8c6fdf4a3ff1d4bd
Nonce 1: 29
Time 1: 22/04/2019 01:46:10.32

Confirming Address 2: d36e2db0175e5f2bb857577db038944829f342a41aebb4757df88f05b382ea82
Confirming Hash 2: 003da711717607f021afc0722fbc13bdf6291b88d294a9b3280bbf8c479020d0
Nonce 2: 3
Time 2: 22/04/2019 01:46:10.32
```



```
Confirms:
Confirmed by: f821c03c9c76e692015dd456f952b66f0fa29547ad5e3be7917ccf8b704aa236
With Hash: 00a37120d52ccebb7cd66c635defc0bc34fe6e840932425cd434872c06593997
At: 22/04/2019 01:46:35.10
```

Facet transaction successfully confirmed, meaning TangleTokens can now be spent.



```
Address: mdG6+BLgzICx4qGg3cyWHab1K/7vivhUpYkQzk80plf4eH74eJTUnjWHhexk5M5KoBEBCh27EnTdD4BzuDiSEA==
Availible Balance: 500
Pending Balance: 500
```

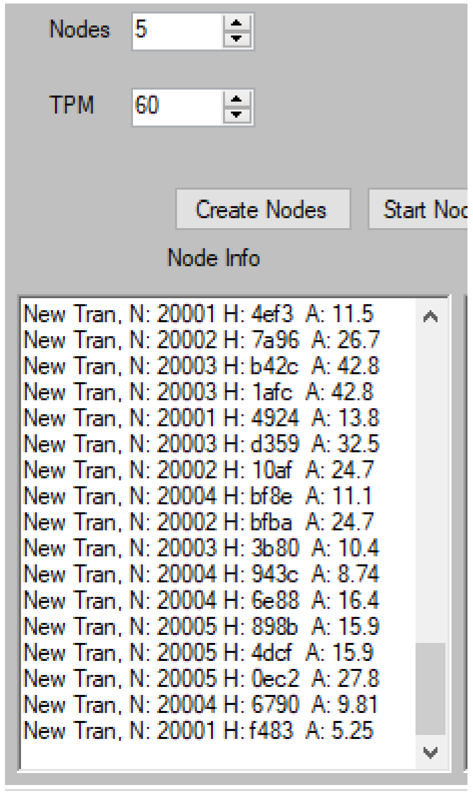
Balance check confirms this.

Test: Ensure node can handle performance of network with high TPM

Steps:

- Generate multiple experimental nodes
- Configure TPM to 60
- Ensure system can handle transactions and work as intended

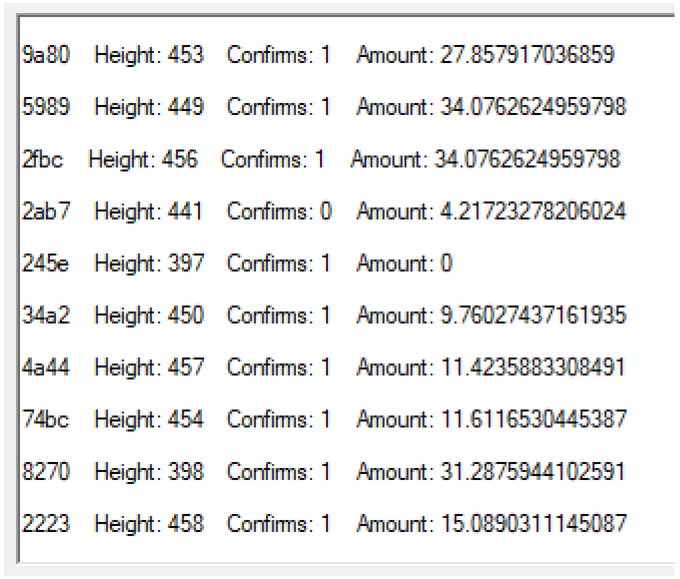
Results:



Nodes 5

TPM 60

```
New Tran, N: 20001 H: 4ef3  A: 11.5
New Tran, N: 20002 H: 7a96  A: 26.7
New Tran, N: 20003 H: b42c  A: 42.8
New Tran, N: 20003 H: 1afc  A: 42.8
New Tran, N: 20001 H: 4924  A: 13.8
New Tran, N: 20003 H: d359  A: 32.5
New Tran, N: 20002 H: 10af  A: 24.7
New Tran, N: 20004 H: bf8e  A: 11.1
New Tran, N: 20002 H: bfba  A: 24.7
New Tran, N: 20003 H: 3b80  A: 10.4
New Tran, N: 20004 H: 943c  A: 8.74
New Tran, N: 20004 H: 6e88  A: 16.4
New Tran, N: 20005 H: 898b  A: 15.9
New Tran, N: 20005 H: 4dcf  A: 15.9
New Tran, N: 20005 H: 0ec2  A: 27.8
New Tran, N: 20004 H: 6790  A: 9.81
New Tran, N: 20001 H: f483  A: 5.25
```



```
9a80   Height: 453   Confirms: 1   Amount: 27.857917036859
5989   Height: 449   Confirms: 1   Amount: 34.0762624959798
2fbc   Height: 456   Confirms: 1   Amount: 34.0762624959798
2ab7   Height: 441   Confirms: 0   Amount: 4.21723278206024
245e   Height: 397   Confirms: 1   Amount: 0
34a2   Height: 450   Confirms: 1   Amount: 9.76027437161935
4a44   Height: 457   Confirms: 1   Amount: 11.4235883308491
74bc   Height: 454   Confirms: 1   Amount: 11.6116530445387
8270   Height: 398   Confirms: 1   Amount: 31.2875944102591
2223   Height: 458   Confirms: 1   Amount: 15.0890311145087
```

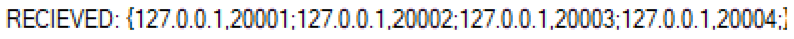
Simple text output shows lots of transactions being generated and are being confirmed in similar fashion to expected functionality.

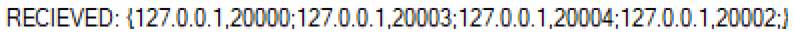
Test: Validate connection pool distributes connections correctly.

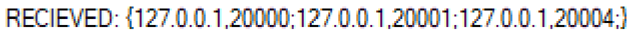
Steps:

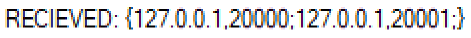
- Open 5 TangleToken clients
- Use the node pool button to ensure all nodes are connected to network

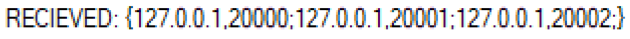
Results:

Node at port 20000: 
```
RECIEVED: {127.0.0.1,20001;127.0.0.1,20002;127.0.0.1,20003;127.0.0.1,20004;}
```

Node at port 20001: 
```
RECIEVED: {127.0.0.1,20000;127.0.0.1,20003;127.0.0.1,20004;127.0.0.1,20002;}
```

Node at port 20002: 
```
RECIEVED: {127.0.0.1,20000;127.0.0.1,20001;127.0.0.1,20004;}
```

Node at port 20003: 
```
RECIEVED: {127.0.0.1,20000;127.0.0.1,20001;}
```

Node at port 20004: 
```
RECIEVED: {127.0.0.1,20000;127.0.0.1,20001;127.0.0.1,20002;}
```

Nodes have various degrees of connectivity to the network, however they all are connected.

6.6.2 Malicious Input Testing

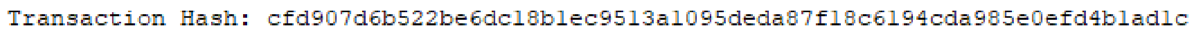
When a malicious input or change is made to either ledger it needs to be identified. It can either be fixed by reporting the issue to another node, or the ledger must refuse to function until it is fixed. Multiple examples of malicious/incorrect inputs will be tested to ensure it can handle them. The experimental nodes are fully automated, so they do not accept inputs, unless the ledger has been changed. This possibility is considered in the following tests.

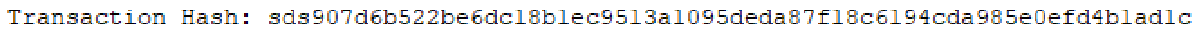
Blockchain

Test: Attempt to edit transaction in ledger

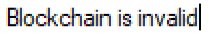
- Blockchain ledger is opened and edited manually
- Hash of transaction is changed and saved
- Blockchain validation methods should catch error

Evidence:


```
Transaction Hash: cfd907d6b522be6dc18b1ec9513a1095deda87f18c6194cda985e0efd4b1ad1c
```


```
Transaction Hash: sds907d6b522be6dc18b1ec9513a1095deda87f18c6194cda985e0efd4b1ad1c
```

Results:

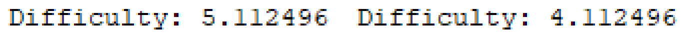

```
Blockchain is invalid
```

Validation mechanism fails to validate transaction, resulting in entire chain being invalid.

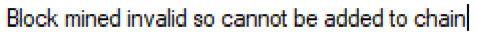
Test: Attempt to edit block data in ledger and mine new block

- Blockchain ledger is opened and edited manually
- Difficulty of block is changed from 5.11 to 4.11 and saved
- A new block is mined, which should be rejected as chain is invalid.

Evidence:


```
Difficulty: 5.112496  Difficulty: 4.112496
```

Results:

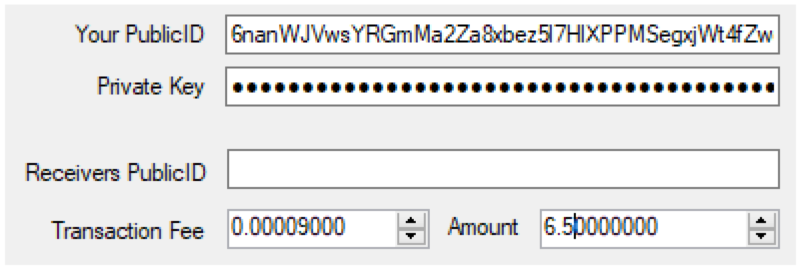

```
Block mined invalid so cannot be added to chain
```

When new block POW is finished the block should be subsequently rejected, as chain is invalid.

Test: Attempting to send transaction with missing info

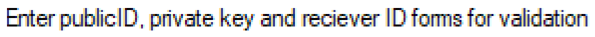
- Generate new wallet
- Attempt to send transaction without filling in receiver ID

Evidence:



Your PublicID	6nanWJVwsYRGmMa2Za8xbez5l7HlXPPMSegxjWt4fZw		
Private Key	●●		
Receivers PublicID			
Transaction Fee	0.00009000	Amount	6.50000000

Results:



Enter publicID, private key and reciever ID forms for validation

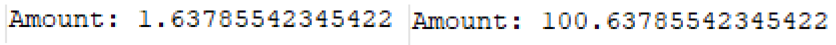
A transaction cannot be generated without a receiver of the funds; therefore, it should be rejected.

Tangle

Test: Attempting to edit amount in ledger

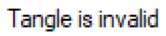
- Tangle ledger is opened and edited manually.
- Amount in a transaction is changed from 1.6 to 100.6 and saved.
- Transaction validation methods should catch this and reject the ledger.

Evidence:



```
Amount: 1.63785542345422 Amount: 100.63785542345422
```

Results:



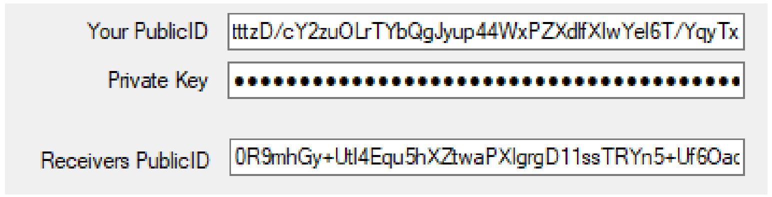
Tangle is invalid

Validation of tangle notices error and rejects the ledger.

Test: Attempt to make transaction with wrong private key

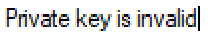
- Generate new wallet
- Change/remove characters in the obscured private key
- Attempt to send a transaction

Evidence (change cannot be seen as private key is obscured):



Your PublicID	tttzD/cY2zuOLrTYbQgJyup44WxPZXdlfXlwYel6T/YqyTx
Private Key	●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●
Receivers PublicID	0R9mhGy+Utl4Equ5hXZtwaPXlgrgD11ssTRYn5+Uf6Oac

Results:



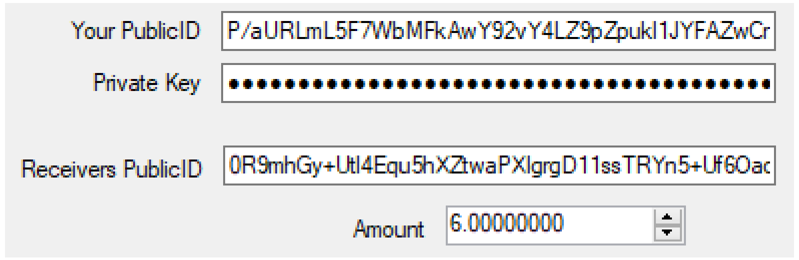
Private key is invalid

Transaction is rejected as private key is invalid

Test: Attempt to make transaction without the balance to support it

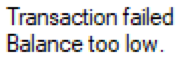
- Generate new wallet
- Send amount higher than current balance
- Attempt to send a transaction

Evidence:



Your PublicID	P/aURLmL5F7WbMFkAwY92vY4LZ9pZpukl1JYFAZwCr
Private Key	●●●
Receivers PublicID	0R9mhGy+Utl4Equ5hXZtwaPXlgrgD11ssTRYn5+Uf6Oac
Amount	6.00000000

Results:



```
Transaction failed
Balance too low.
```

Transaction is rejected as balance is too low.

These tests ensure that both of the systems work as expected. Valid inputs are accepted, and the expected output is then generated. Malicious inputs are caught, invalidating the ledger and prevent the node from operating. The standard input testing not only highlighted that the validation and verification methods work as expected, but also validates the system and its functions. The outputs in these tests require given input data to be manipulated correctly and came out as expected.

However, there is a limitation to the validity of the testing. If the verification methods themselves have been implemented wrong or have bugs; then some malicious inputs could remain undetected or the process could be invalid. Despite this concern the system workflow works as expected, so there is no indication that these methods no not work.

7. The Experiment

7.1 Aims

It is the aim of this experiment to validate if the performance statistics found in the literature survey of the Blockchain and Tangle are well founded. While this experiment shouldn't verify the exact claims, it should instead provide qualitative data about their performance.

Furthermore, it is hypothesized that both systems perform best under certain conditions. Where 'best' can be defined as an optimal amount of energy per transaction and time per transaction. Finding this exact sweet spot is not necessary, but instead it should be an aim of this experiment to find the preferable and undesirable conditions.

Trustless ledgers have still not been accepted by the world as a legitimate payment option. Electronic payments are favoured instead, as the trustless aspect does not outweigh the negatives. In the case of full-scale adoption many transactions will need to be confirmed per second. So, it is also an aim of this experiment to verify the most suitable ledger for the future of trustless ledgers.

7.2 Method

The parameters of the experiment can have a large influence in the results. Therefore, a variety of tests will be made, to understand how the systems reacts in different scenarios.

In theory the Tangle has a scaling performance that results in a higher TPS when more transactions are being generated. However, in a low transaction environment the Tangle may struggle. Additionally, as the total hash power of a Blockchain network scales to the difficulty of mining, a variance in the hashing power should be tested to confirm this.

Each test will be performed in a fair environment, control variables will be kept static and all tests will be performed on the same PC.

Independent Variables (Varying inputs):

- Transactions generated per minute
- CPU power allocated (for Blockchain)
- The ledger type used – Blockchain/Tangle

The CPU power allocated can be considered the hashing power of the network. It is important to observe a change in difficulty and hashes per transaction in Blockchain with a throttled network hashing power.

Dependent Variables (Measure outputs):

- Hashes per Transaction
- Time per confirmation of transaction
- Time per 5 confirmations (merchant confirmation definition, on Blockchain only)

Control Variables:

- Number of nodes running (10)
- Each node will start from a fresh ledger
- Each test will last 15 minutes
- The hardware running the experiment (Intel Core i7-6700k CPU 4GHz and 250 GB SSD)

To satisfy these aims, three tests with varying transaction rates will be executed. These tests will be ran on both ledgers and also an additional time on Blockchain where the hashing power of the network is throttled. These three tests can be classified individually as high (60), medium (15) and low (5) transaction rates per minute.

7.3 Results

The results for each experiment are listed below. The full experimental results, including the graphs derived by the experimental harness, are included in the appendices of this report.

Test 1 (Low transaction rate: 5 TPM):

TestCoin Results:

Ledger Type	Blockchain
Average hashes/transaction	1,021,816
Average time till added to chain (Seconds)	10.3
Average time for 5 chain confirmations (Seconds)	111

TestCoin (Throttled) Results:

Ledger Type	Blockchain (Throttled)
Average hashes/transaction	170,053
Average time till added to chain (Seconds)	17.6
Average time for 5 chain confirmations (Seconds)	163

TangleToken Results:

Ledger Type	Tangle
Average hashes/transaction	464
Average time for 1 confirmation	14.5

Test 2 (Medium transaction rate: 15 TPM):

TestCoin Results:

Ledger Type	**Blockchain**
Average hashes/transaction	548,698
Average time till added to chain (Seconds)	22.7
Average time for 5 chain confirmations (Seconds)	76.1

TestCoin (Throttled) Results:

Ledger Type	**Blockchain (Throttled)**
Average hashes/transaction	73,645
Average time till added to chain (Seconds)	63.8
Average time for 5 chain confirmations (Seconds)	178

TangleToken Results:

Ledger Type	**Tangle**
Average hashes/transaction	515
Average time for 1 confirmation	8.39

Test 3 (High transaction rate: 60 TPM):

TestCoin at full CPU allocation with a high transaction rate is overloaded. At the CPU is always occupied mining, while generating lots of transactions that need to be shared across the network, the system is incredibly slow and unstable. Unfortunately, it is not possible to run TestCoin for 15 minutes with 60 TPM. However with throttled mining, TestCoin can run at 60 TPM as more CPU is available.

TestCoin (Throttled) Results:

Ledger Type	Blockchain (Throttled)
Average hashes/transaction	34,281
Average time till added to chain (Seconds)	85.7
Average time for 5 chain confirmations (Seconds)	218

TangleToken Results:

Ledger Type	Tangle
Average hashes/transaction	504
Average time for 1 confirmation	5.67

7.4 Limits of validity

These results can only be considered an indication of the performance of the ledgers. The data cannot be considered quantitative due to the limitations of the system and the hardware. The hashing power of a single CPU is minute in comparison to the hashing power of a real network of distributed nodes. Even a single node in the network will have a significantly higher amount hashing power then a single CPU, due to the specialised hardware in a mining node. Without this untenable hashing power, a real indication of the number of hashes required per transaction can only be speculated upon using these results. The scaling difficulty element in the TestCoin system suggests the hashes per transactions will be much higher but cannot provide any definitive results.

Alongside hashing power, the number of nodes on the network has been reduced. Given the limitations of the system and the hardware, it becomes infeasible to host more than 10-20 nodes on a single computer. A larger network wouldn’t appear to affect the hashes per transaction or time per transaction but without a way to verify this claim, you cannot dismiss this factor. In addition, the network is local, meaning factors such as latency and disconnections are not possible. While this would appear to be a benefit as it means the variables are static throughout the experiment, it discards any nuances you could expect in a real distributed network.

The implementation is not perfect either, given the time limitations neither implementation can be as complex or polished as Bitcoin or IOTA. TestCoin and TangleToken are good representations of Blockchain and Tangle but lack some of the elements real trustless ledgers use. IOTA for example uses centralised nodes to distribute checkpoints on the Tangle. This is not present on TangleToken as it undermines the trustless aspect. However, this difference alone results in a significantly different approach in confirming transactions. Although the Tangle results should still remain useful for generic implementations of the Tangle, opposed to IOTA.

8. Discussion and Conclusion

8.1 Results Analysis

Blockchain

In the Blockchain experiment, a clear downwards trend is visible regarding hashes per transaction. As TPM increased, the number of hashes per transaction decreased. Blocks have a max capacity of 15 transactions, so the closer to full, the better the distribution of hashes/transactions. By 60 TPM the average number of hashes per transaction was at its lowest; 34281. In this same experiment, if every block was full of transactions, the average number of hashes per transaction would be 31095, as shown in appendix C. This proves that at 60 TPM blocks were reaching full capacity.

The number of hashes per transaction at a throttled full capacity rate are reasonable. However, in the experiments with low TPM or full CPU, the number of hashes per transactions was significantly higher. At 5 TPM with full CPU, over 1 million hashes were used per transaction. The is reveals two clear weaknesses in Blockchain in regard to hashes/transaction. A low TPM rate, results in an unnecessarily large amount of work being done to add a few transactions to the chain. If blocks are not at full capacity, energy is wasted. Compounding this with the scaling factor of difficulty, reveals that an untenable hashing power and therefore energy, will be wasted on blocks that are not full. This is more than speculation too, as Bitcoin and Ethereum frequently create blocks that are not at full capacity [34] [88].

There are two definitions to ‘confirming’ a transaction in Blockchain; a technical confirm, which occurs when a transaction is added to the chain, and a merchant confirm. A merchant confirm is defined as ***x*** blocks being added to the chain after the transaction in focus has been added. In this experiment ***x*** is set to 5. This is largely an arbitrary number, as this decision is normally made in regard to security. However, given the limited time per experiment, this number should be reasonably low.

Both definitions of confirm appear to trend upwards in respect to time as the TPM is increased. This upwards trend is apparent in both the full CPU and throttled TestCoin experiments. As more and more transactions are added to the pool, it is apparent that they must wait longer before they can be confirmed/added to chain. A larger block size will allow help to improve this issue, but it can only be increased by a limited amount before performance and security issues arise.

The TestCoin transaction pool can only store a finite number of transactions at once, set to 100 in this experiment. Once this number has been met, any new transactions take the place of an old one. Normally the transaction with the lowest fee (lowest benefit to the miner) is removed. There is no mechanism to count the number of removed transactions, as each node has its own pool. But it is completely possible that some transactions were lost, especially if the block rate was lower expected.

Tangle

The Tangle had little variance in the average number of hashes per transaction throughout the experiments; which is unsurprising as the difficulty of POW stayed constant throughout, by design. As transactions only required around 500 hashes each, they could be almost instantly added to the Tangle. This result is predictable but can be used as a comparison point to Blockchain's hashes/transaction.

In regard to the Tangle, the focus point of this experiment is the time it takes for a transaction to receive a single confirm after generation. It is expected that the number transactions generated per second directly correlates to the average time for a transaction to be confirmed. A fast transaction confirmation rate is preferable.

A clear trend can be seen in the results, backing up this correlation. As the TPM is increased the average confirmation time reduces. Results show that at 5 TPM there is an average confirmation time of 14.5 seconds, at 15 TPM it is 8.39 seconds and at 60 TPM the rate is 5.67 seconds. This supports the hypothesis, and also implies there may be diminishing returns at higher rates. Although this may be due to limitations in the system, as at a high transaction rate the system may loss performance regarding tip selection; meaning unconfirmed transactions are not selected as tips as often as expected.

At the end of each TangleToken experiment the number of unconfirmed transactions is tallied. Given there is an element of randomness in tip selection, there is a chance a transaction will not get selected. The Monte Carlo sampling prefers to pick transactions that have the highest depth, therefore the longer a transaction goes unconfirmed the less likely it stands to be picked. In the extended results at appendix C, the number and percentage of unconfirmed transactions are stated. At 5, 15 and 60 TPM, unconfirmed transactions consist of 10%, 15% and 20% of all the transactions generated during the experiment respectively. The remaining transactions have at least one confirm. It is admittedly hard to conclude if this discrepancy is a result of the TPM or the demand on the system. Regardless, it is still a possibility under any TPM that some transactions may never be confirmed.

8.2 Comparison and Discussion

There is almost no contest in respect to energy used. In a low CPU Blockchain, the Tangle still makes substantially less hashes per transaction. In a low/medium hashing power network this difference is not too imperative, however in a system with lots of nodes and hashing power a Tangle ledger appears drastically preferable.

TangleToken appeared to almost outperform TestCoin from the start. Transactions were confirmed faster on Tangle than they were added to the chain on Blockchain, in all but 5 TPM. It is clear that the performance of the Tangle struggles in a low TPS network, but it quickly matched Blockchain in the 15 TPM tests. It began at excel at 60 TPM, reaching confirmation speeds quicker than TestCoin was even capable of. In the meantime, TestCoin struggled, resulting in the slowest confirmation times measured.

The best use-case of a Blockchain would involve a network with a strong, but not untenable hashing power. It would be strong enough to not be vulnerable to 51% attacks, but not be wasteful. The transaction rate should be high enough that little to none of the capacity of a block is left over, with minimal transactions left in the pool. The exact numbers for this scenario would depend on the implementation itself. But it is clear that Blockchain it its current form has finite caps, restricting its domain of strong performance.

As hypothesised the Tangle performs better at a higher TPS. The claim that it scales infinitely, getting faster and faster as TPS increases cannot be proven in this experiment. The system and available computing power restricted the ability to test this claim. What can be said however, is that the Tangle proved that this claim is worthy of further investigation.

Without being able to verify that the Tangle thrives with an exponential TPS, the perfect Tangle use-case cannot be defined fully. Although in a real-world full adoption scenario, it would be fair to assume that Tangle would be the best option. In this scenario a trustless ledger will need to match Visa's 45000 transactions per second [90]. And based off the evidence in this report, Tangle is the only candidate.

8.3 Social, Ethical, Legal and Health and Safety Issues

Trustless ledgers is a branch of computational science that is surprisingly associated with a platter of issues. This is partly responsible to the rise in popularity as cryptocurrencies as an investment opportunity. In its current state investing in crypto is highly speculative and very close to gambling. It is not unheard of that people have taken out loans or invested their life savings in cryptocurrencies. This report should not be considered sound investment advice and is by no means a prediction of the future of cryptocurrencies.

Cryptocurrency exchanges are still largely unregulated, which enables highly unethical tactics such as wash trading. These tactics are illegal in stock trading as cryptocurrencies are still in an inception period. In the meantime, trading should be approached with caution.

As Cryptocurrencies are digital assets, protected by a public and private key, it is highly recommended to keep the private key as secure as possible. It is the target of hackers and criminals to seize users' private keys to steal their funds. Funds should be stored on a cold wallet, and away from exchanges, which have a reputation for being heisted [91]. In addition, as cryptocurrencies are somewhat anonymous and hard to track (in particular Monero), illegal behaviour is enabled. Anonymous trading makes tax avoidance, money laundering and black markets possible.

8.4 Conclusion and Future Work

As targeted, two experimental models have been designed that both respectively demonstrate a full Blockchain and Tangle system. This has been achieved in accordance with the design principles and replicate similar functionality to real implementations such as Bitcoin or IOTA. The experimental harness successfully generated meaningful results that are reasonably decisive such that a useful conclusion can be derived.

Both systems demonstrated that trustless ledgers are a viable option for storing transactions. Testing proved their security, irreversibility and immutability and validated the systems operated as expected. While both models worked as expected it was apparent that the Tangle used substantially less energy (hashes) per transaction in all cases. Transaction confirmation time varied with the independent variables with both ledgers. On a small unpopular network, the Tangle is slow and insecure. On a popular network the Tangle significantly out performs Blockchain.

There are many subtleties that need to be considered when selecting a trustless ledger. Despite the Tangles benefits, Blockchain is still considered to be the true trustless and secure model. Current Tangle implementations such as IOTA use centralised components to mitigate any security risks. This sacrifices the trustless aspect; and in a scenario where a secure trustless implementation is intrinsic, a Blockchain will likely be preferred. The price of unconfirmed transactions on the Tangle needs to be considered too. Tangle cannot guarantee that every transaction will ever be confirmed; while Blockchain may take a long time but ensures that all transactions are added to the chain (given a large transaction pool).

The burden of Proof of Work is clear, and it would seem that large POW systems are endangered by their extreme energy usage. The Tangle stands as a candidate to take its place, but other solutions do exist. Other proof protocols exist such as Proof of Stake, where a node which holds a large proportion of the currency can vote on the next block added to the chain. This approach uses much less energy, but significantly undermines the security of the chain as a voting node could choose to operate illegally. Perhaps Tangle is the best compromise, as reduces the energy used substantially and does not completely undermine the trustless aspect.

It should be mentioned that elements of some Blockchain implementations could not be featured in this report due to scope and time issues. In the past few years Bitcoin added a ‘Lightning network’ protocol. It is a complex mechanism that allows multiple offline transactions which are all encompassed with one transaction, so the fee only has to be payed once. This absence should not influence the results too much but is worthy of consideration given additional time.

With additional time this system could be repurposed as an educational tool. In its current state, the system is reasonably simple to use but could be expanded upon to visualise the nuances of each function. There is an absence of Blockchain and Tangle knowledge, even in cryptocurrencies communities, where investment ‘knowledge’ holds higher value than knowledge about the technology itself. Education on the subject could create mature views in regard to the future of cryptocurrency and its possible adoption.

Alongside fulfilling the requirements of an experimental testbed, both implementations can be used independently as a trustless ledger. Hypothetically these systems could be used a real trustless ledgers distributed across the world, given that the local system blockade is removed. This would only require a small amount of additional development, as the majority codebase would not need changing.

9. Reflection

9.1 On the Project

The scope of this project was huge, once the solution had been identified I realised that the amount of development effort required was substantial. With a large amount of development work required, a large amount of research and documentation was also necessary. As a result, this project has a lot of content, at the cost of a lot of time. Although I believe a simpler approach would result in less useful experimental data, or a non-functional model. With a chance to be more conscious of the impending work, I should have invested time my time better at the beginning of the project. Development was time consuming, and exerted pressure on other coursework deadlines and the report too. A chance to start development earlier could have eased the trouble caused nearing the end of the deadline.

Throughout development, performance became a concern. As more read/write operations became prevalent to ensure the program was consistent, the longer processes took. This became relevant when hosting multiple nodes, as all perform many write/read operations. When compounding this with the many network messages sent between nodes during testing, performance began to drop. When using all the CPU (for Blockchain testing) this restricted the number of operations and nodes that could be ran, meaning at full CPU Blockchain test could not be ran at 60 TPM. I attempted multiple optimisations to reduce this CPU load throughout the project, which did help enable high transaction tests on the Tangle and low CPU Blockchain, but it was not enough for full CPU Blockchain. It is tricky to be sure if this performance issue is a result of a poor system implementation, or if it is a limit of the hardware; as there are no other experimental Blockchain/Tangle tools to test this issue on.

9.2 Personal

The inspiration for this project came from the personal ambition to learn about Blockchain and Tangle systems. Due to prior exposure to the subject I was keen to design my final year project around this topic. Energy consumption and transaction time are perhaps the two largest areas of contestation in the crypto community. An investigation into this area has a real purpose, and also supports my ambition to learn.

Personally, I feel the project went very well. I learned extensive amounts about Blockchain and Tangle technologies, and also managed to expand to other areas of study; such as networking and data analysis. I am happy with the final result of the project and report. I feel it covers all the content I originally targeted.

Planning and designing this project was a chance to learn a lot in areas I didn’t feel confident about. I was excited to start developing but weary over designing the system to fulfil the problem. Fortunately, my supervisor provided a lot of advice into appropriately designing a system. I was able to borrow books on UML and design a Goal Sketching model with my supervisor’s aid. Retrospectively my only change regarding the design would be working on it earlier, as I unnecessarily stalled it as I lacked confidence.

References

[1] LLFOURN, “A Brief History of Ledgers,” 15 Febuary 2018. [Online]. Available: https://medium.com/unraveling-the-ouroboros/a-brief-history-of-ledgers-b6ab84a7ff41.

[2] P. D. a. R. K. E. A. B. Hans J. Nissen, “A Short History of Proto-Cuneiform,” CDLI Wiki, 1993. [Online]. Available: http://cdli.ox.ac.uk/wiki/doku.php?id=proto-cuneiform. [Accessed 29 3 2019].

[3] R. K., “Archival view of P000735,” cdli, 1994. [Online]. Available: https://cdli.ucla.edu/search/archival_view.php?ObjectID=P000735. [Accessed 29 3 2019].

[4] D. O'Brien, “The Development of Monetary Economics,” Edward Elgar Publishing Limited, Cheltenham, 2007.

[5] A. Hayes, “Double Entry Definition,” Investopedia, 8 Febuary 2019. [Online]. Available: https://www.investopedia.com/terms/d/double-entry.asp. [Accessed 29 3 2019].

[6] Silk Road, “Chinese paper money,” Silk Road, [Online]. Available: http://www.silk-road.com/artl/papermoney.shtml. [Accessed 29 3 2019].

[7] Computer Smiths, “History of Chinese Invention - The Invention of Paper Money,” Computer Smiths, [Online]. Available: https://www.computersmiths.com/chineseinvention/papermoney.htm. [Accessed 29 3 2019].

[8] Higher Rock Education, “Commodity Money,” High Rock Education and Learning Inc., [Online]. Available: https://www.higherrockeducation.org/glossary-of-terms/commodity-money. [Accessed 29 3 2019].

[9] J. Hall, “Fiat Currency: What It Is and Why It's Better Than a Gold Standard,” The Motley Fool, 6 12 2015. [Online]. Available: https://www.fool.com/investing/general/2015/12/06/fiat-currency-what-it-is-and-why-its-better-than-a.aspx. [Accessed 29 3 2019].

[10] IG, “Fiat Currency definition,” IG, [Online]. Available: https://www.ig.com/uk/glossary-trading-terms/fiat-currency-definition. [Accessed 29 3 2019].

[11] A. M. Research, “Contactless Payments Market to Reach $27.23 Bn, Globally, by 2023 at 20.8% CAGR: Allied Market Research,” Cision, 7 2 2019. [Online]. Available: https://www.prnewswire.com/news-releases/contactless-payments-market-to-reach-27-23-bn-globally-by-2023-at-20-8-cagr-allied-market-research-300791619.html. [Accessed 29 3 2019].

[12] E. C. Bank, *Press Release; Payments statictics for 2016,* 2017.

[13] Dr.Econ, “Do all banks hold reserves, and, if so, where do they hold them?,” FEDERAL RESERVE BANK OF SAN FRANCISCO, November 2001. [Online]. Available: https://www.frbsf.org/education/publications/doctor-econ/2001/november/bank-reserve-requirements/. [Accessed 29 3 2019].

[14] H. Tomlinson, “Banks run out of money as Indians hoard cash,” The Times, 20 April 2018. [Online]. Available: https://www.thetimes.co.uk/article/banks-run-out-of-money-as-indians-hoard-cash-again-rlvlz6fn5 . [Accessed 29 3 2019].

[15] Money Super Market, “How the Financial Services Compensation Scheme protects your savings,” Money Super Market, 19 December 2018. [Online]. Available: https://www.moneysupermarket.com/savings/protecting-your-savings-guide/. [Accessed 29 3 2019].

[16] UK Finance, “Fraud the Facts 2018,” UK Finance, 2018.

[17] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[18] A. R. Sorkin, “Lehman Files for Bankruptcy; Merrill Is Sold,” The New York Times, 14 September 2008. [Online]. Available: https://www.nytimes.com/2008/09/15/business/15lehman.html. [Accessed 10 December 2018].

[19] B. Chu, “Financial crisis 2008: How Lehman Brothers helped cause 'the worst financial crisis in history',” Independent, 12 September 2018. [Online]. Available: https://www.independent.co.uk/news/business/analysis-and-features/financial-crisis-2008-why-lehman-brothers-what-happened-10-years-anniversary-a8531581.html . [Accessed 25 October 2018].

[20] M. Palframan, “Ten years after the financial crisis - two thirds of British people don't trust banks,” YouGov, 2019 August 2018. [Online]. Available: https://yougov.co.uk/topics/finance/articles-reports/2018/08/29/ten-years-after-financial-crisis-two-thirds-britis. [Accessed 25 October 2018].

[21] V. Morabito, “Business Innovation Through Blockchain,” Springer, Milan, 2017.

[22] CoinMarketCap, “Global Charts; Total Market Capitalization,” CoinMarketCap, [Online]. Available: https://coinmarketcap.com/charts/. [Accessed 29 March 2019].

[23] B. Weber, “The Millennial Gold Standard: Bitcoin,” Medium, 31 July 2018. [Online]. Available: https://medium.com/dialogue-and-discourse/the-millennial-gold-standard-bitcoin-c29857af8fd7. [Accessed 29 March 2019].

[24] S. O'Neal, “Corrupt Governance? What We Know About Recent EOS Scandal,” CoinTelegraph, 5 October 2018. [Online]. Available: https://cointelegraph.com/news/corrupt-governance-what-we-know-about-recent-eos-scandal. [Accessed 29 March 2019].

[25] D. Canellis, “Here’s why Bitcoin’s blockchain has blocks that go over the 1MB limit,” The Next Web, 2019. [Online]. Available: https://thenextweb.com/hardfork/2018/07/12/bitcoin-block-size/ . [Accessed 30 March 2019].

[26] Digiconomist, “Bitcoin Energy Consumption Index,” Digiconomist, [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption. [Accessed 10 December 2018].

[27] UKPower, “Gas & Electricity Tariff Prices per kWh,” UKPower, 2019. [Online]. Available: https://www.ukpower.co.uk/home_energy/tariffs-per-unit-kwh. [Accessed 30 March 2019].

[28] Bitcoin, “Developer Documentation,” Bitcoin, 2019. [Online]. Available: https://bitcoin.org/en/developer-documentation. [Accessed 31 March 2019].

[29] Blockchain, “Block #570092,” BLOCKCHAIN LUXEMBOURG S.A, 3 April 2019. [Online]. Available: https://www.blockchain.com/btc/block/0000000000000000000026960d36e9ffe255e4bde8656a843cea2f32612b1f4b12. [Accessed 3 April 2019].

[30] D. G. Wood, “ETHEREUM: A SECURE DECENTRALISED GENERALISED TRASNACTION LEDGER,” 2015.

[31] Bitcoin Wiki, “Genesis Block,” Bitcoin Wiki, [Online]. Available: https://en.bitcoin.it/wiki/Genesis_block. [Accessed 4 April 2019].

[32] J.-S. Coron, Y. Dodis, C. Malinaud and P. Puniya, “Merkle-Damg˚ard Revisited: How to Construct a Hash Function,” Luxembourg.

[33] P. R. a. T. Shrimpton, “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance,” 2004.

[34] Blockchain, “Block #570100,” BLOCKCHAIN LUXEMBOURG S.A, 4 April 2019. [Online]. Available: https://www.blockchain.com/btc/block/00000000000000000002e89c18ce9c2ee2c870fde7f0955918fea1e229885048. [Accessed 4 April 2019].

[35] Blockgeeks, “What Is Hashing? Under The Hood Of Blockchain,” Blockgeeks, 2017. [Online]. Available: https://blockgeeks.com/guides/what-is-hashing/. [Accessed 4 April 2019].

[36] S. Ray, “Merkle Trees,” Hackernoon, 15 December 2017. [Online]. Available: https://hackernoon.com/merkle-trees-181cb4bc30b4. [Accessed 4 April 2019].

[37] B. Garner, “Merkle Tree Hashing: How Blockchain Verification Works,” Coin Central, 3 September 2019. [Online]. Available: https://coincentral.com/merkle-tree-hashing-blockchain/. [Accessed 4 April 2019].

[38] K. M. J. K. Alex Chumbley, “Merkle Tree,” Brilliant, [Online]. Available: https://brilliant.org/wiki/merkle-tree/. [Accessed 4 April 2019].

[39] J. d. Koning, “Internet of Coins Hybrid Assets for Peer-to-Peer Intersystemic Value Transfer,” 2014.

[40] Bitcoinfees, “Unconfirmed Transactions/ Transactions Today,” Bitcoinfees, 2019. [Online]. Available: https://bitcoinfees.earn.com/. [Accessed 5 April 2019].

[41] Coinbase, “Why is my transaction 'Pending'?,” Coinbase, [Online]. Available: https://support.coinbase.com/customer/en/portal/articles/593836-why-is-my-transaction-pending-. [Accessed 5 April 2019].

[42] D. Liu and L. J. Camp, “Proof of Work can Work,” Indiana University, Indiana, 2006.

[43] Etherchain, “Average block time of the Ethereum network,” Etherchain, [Online]. Available: https://www.etherchain.org/charts/blockTime. [Accessed 5 April 2019].

[44] Bitcoin talk, “Nonce and extranonce algorithm,” 28 April 2015. [Online]. Available: https://bitcointalk.org/index.php?topic=1040859.0. [Accessed 5 April 2019].

[45] Bitcoin Wiki, “Controlled Supply,” Bitcoin Wiki, [Online]. Available: https://en.bitcoin.it/wiki/Controlled_supply. [Accessed 5 April 2019].

[46] S.-J. Chung and E. Lee, “In-Depth Analysis of Bitcoin Mining Algorithm Across Different Hardware,” Carnegie Mellon University, Pittsburgh.

[47] J. Frankenfield, “Soft Fork,” Investopedia, 6 Febuary 2019. [Online]. Available: https://www.investopedia.com/terms/s/soft-fork.asp. [Accessed 5 April 2019].

[48] N. Acheson, “Hard Fork vs Soft Fork,” Coindesk, 16 March 2018. [Online]. Available: https://www.coindesk.com/information/hard-fork-vs-soft-fork. [Accessed 5 April 2019].

[49] A. Gervais, G. O.Karame, K. Wust, V. Glykantzis, H. Ritzdorf and S. Capkun, “On the Security and Performance of Proof of Work”.

[50] D. C. Investor, “Bitcoin forks 2017 Illustration,” Reddit, 2017. [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/7a9y3h/bitcoin_forks_2017_illustration/. [Accessed 5 April 2019].

[51] R. Versteegden, “How Blockchain ensures trasnsactions are not lost?,” StackExchange, 27 December 2017. [Online]. Available: https://bitcoin.stackexchange.com/questions/66736/how-blockchain-ensures-transactions-are-not-lost. [Accessed 5 April 2019].

[52] S. Khatwani, “How Long Does It Take To Transfer Bitcoins And Why?,” CoinSutra, 8 April 2018. [Online]. Available: https://coinsutra.com/bitcoin-transfer-time/. [Accessed 5 April 2019].

[53] J. Vermeulen, “Bitcoin and Ethereum vs Visa and PayPal – Transactions per second,” My Broadband, 22 April 2017. [Online]. Available: https://mybroadband.co.za/news/banking/206742-bitcoin-and-ethereum-vs-visa-and-paypal-transactions-per-second.html. [Accessed 5 April 2019].

[54] J. Frankenfield, “Double-Spending,” Investopedia, 5 July 2018. [Online]. Available: https://www.investopedia.com/terms/d/doublespending.asp. [Accessed 5 April 2019].

[55] Y. Gola, “Bitcoin Cash: Single Mining Pool Controls 50% of Hashrate,” CCN, 1 March 2019. [Online]. Available: https://www.ccn.com/bitcoin-cash-single-mining-pool-controls-half-of-hashrate. [Accessed 5 April 2019].

[56] G.F., “Why bitcoin uses so much energy,” The Ecnomist, 9 July 2019. [Online]. Available: https://www.economist.com/the-economist-explains/2018/07/09/why-bitcoin-uses-so-much-energy. [Accessed 5 April 2019].

[57] CoinMarketCap, “Bitcoin,” CoinMarketCap, [Online]. Available: https://coinmarketcap.com/currencies/bitcoin/. [Accessed 5 April 2019].

[58] W. Peaster, “Dark Web Dealer Pleads Guilty After Being Tracked Through Bitcoin Addresses,” BitsOnline, 12 June 2018. [Online]. Available: https://bitsonline.com/dark-web-dealer-bitcoin-pleads-guilty/. [Accessed 5 April 2019].

[59] N. v. Saberhagen, “CryptoNote v 2.0,” 2013.

[60] E. Faggart, “What Happens to Bitcoin Miners When all Coins are Mined?,” news.Bitcoin.com, 15 August 2015. [Online]. Available: https://news.bitcoin.com/what-happens-bitcoin-miners-all-coins-mined/. [Accessed 5 April 2019].

[61] S. Popov, “The Tangle,” 2018.

[62] C. Cachin and M. Vukolic, “Blockchain Consensus Protocols in the Wild,” IBM Research, Zurich, 2017.

[63] M. Rosulek, “How IOTA makes bright future for Internet of Things,” Medium, 18 June 2017. [Online]. Available: https://medium.com/@MartinRosulek/how-iota-makes-future-for-internet-of-things-af14fd77d2a3. [Accessed 6 April 2019].

[64] Zauz, “How to start a new offline subtangle,” StackExchange, 15 Febuary 2018. [Online]. Available: https://iota.stackexchange.com/questions/1523/how-to-start-a-new-offline-subtangle. [Accessed 6 April 2019].

[65] Q. Bramas, “The Stability and the Security of the Tangle,” 2018.

[66] I. Fiedler, “The Distribution of IOTA Tokens,” Medium, 7 March 2017. [Online]. Available: https://medium.com/@IngoFiedler_96862/the-distribution-of-iota-tokens-dc70ea49b148. [Accessed 6 April 2019].

[67] Zuaz, “How exactly are transactions verified and confirmed in IOTA,” StackExchange, 21 Febuary 2018. [Online]. Available: https://iota.stackexchange.com/questions/1577/how-exactly-are-transactions-verified-and-confirmed-in-iota. [Accessed 6 April 2019].

[68] J. Zhang, “Why the Crypto Dream Does Not Work, For Now.,” Hackernoon, 5 Auguest 2018. [Online]. Available: https://hackernoon.com/why-the-crypto-dream-does-not-work-for-now-f561fc21784. [Accessed 6 April 2019].

[69] M. Beedham, “Moonday Mornings: Dark Web Bitcoin transactions doubled in 2018,” The Next Web, 21 January 2019. [Online]. Available: https://thenextweb.com/hardfork/2019/01/21/moonday-dark-web-bitcoin-transactions-double/. [Accessed 5 April 2019].

[70] B. Schneier, “THERE'S NO GOOD REASON TO TRUST BLOCKCHAIN TECHNOLOGY,” Wired, 6 Febuary 2019. [Online]. Available: https://www.wired.com/story/theres-no-good-reason-to-trust-blockchain-technology/. [Accessed 6 April 2019].

[71] A. Frolova, “Will Crypto and Regulators Ever Be Friends?,” Medium, 1 May 2018. [Online]. Available: https://medium.com/exodus-movement/will-crypto-and-regulators-ever-be-friends-8d9d9edbbf82. [Accessed 6 April 2019].

[72] S. Das, “Breaking: China’s Central Bank Bans all ICOs,” CCN, 9 April 2017. [Online]. Available: https://www.ccn.com/breaking-chinas-central-bank-bans-icos. [Accessed 6 April 2019].

[73] D. Stricker, Twitter, 21 June 2018. [Online]. Available: https://twitter.com/dantherealm4n/status/1009847828259143680 . [Accessed 30 October 2018].

[74] Digiconomist, “Ethereum energy consumption,” Digiconomist, [Online]. Available: https://digiconomist.net/ethereum-energy-consumption. [Accessed 7 April 2019].

[75] Blockchain , “Unconfirmed Transactions Live updating list of new bitcoin transactions,” Blockchain, [Online]. Available: https://www.blockchain.com/btc/unconfirmed-transactions. [Accessed 7 April 2019].

[76] “The Tangle main net,” [Online]. Available: http://tangle.glumb.de/. [Accessed 7 April 2019].

[77] Etherscan, “Ethereum Blockchain explorer,” Etherscan, [Online]. Available: https://etherscan.io/. [Accessed 7 April 2019].

[78] cdecker, “With 100% segwit transactions, what would be the max number of transaction confirmation possible on a block?,” StackExchange, 13 September 2017. [Online]. Available: https://www.quora.com/What-is-the-maximum-number-of-transactions-per-second-that-the-Bitcoin-network-can-handle. [Accessed 7 April 2019].

[79] J. MacAvoy, “IOTA : A Cryptocurrency With Infinite Scalability And No Fees,” Interesting Engineering, 7 December 2017. [Online]. Available: https://interestingengineering.com/iota-a-cryptocurrency-with-infinite-scalability-and-no-fees. [Accessed 7 April 2019].

[80] CPereez19, “How many transactions per second can Ethereum currently handle? What changes will allow the network to be able to handle more? [duplicate],” StackExchange, 25 May 2018. [Online]. Available: https://ethereum.stackexchange.com/questions/49484/how-many-transactions-per-second-can-ethereum-currently-handle-what-changes-wil/49488. [Accessed 7 April 2019].

[81] Bitnodes, “GLOBAL BITCOIN NODES DISTRIBUTION,” Bitnodes, [Online]. Available: https://bitnodes.earn.com/. [Accessed 7 April 2019].

[82] Ethernodes, “Ethereum Mainnet Network,” Ethernodes, [Online]. Available: https://www.ethernodes.org/network/1. [Accessed 7 April 2019].

[83] Blockchain, “Hash Rate,” Blockchain, [Online]. Available: https://www.blockchain.com/en/charts/hash-rate. [Accessed 7 April 2019].

[84] Ethstats, Ethstats, [Online]. Available: https://ethstats.net/. [Accessed 7 April 2019].

[85] IOTA, “IOTA sourcecode,” Github, [Online]. Available: https://github.com/iotaledger. [Accessed 28 April 2019].

[86] Google, “Bitcoin Wallet,” Bitcoin Wallet Developers, [Online]. Available: https://play.google.com/store/apps/details?id=de.schildbach.wallet. [Accessed 10 April 2019].

[87] SinfulPhilanthropist, “What's the hashrate for an average gaming computer?,” Reddit, [Online]. Available: https://www.reddit.com/r/dogecoin/comments/1valv1/whats_the_hashrate_for_an_average_gaming_computer/. [Accessed 10 April 2019].

[88] Etherscan, “Ethereum Block Explorer,” Etherscan, [Online]. Available: https://etherscan.io/. [Accessed 10 April 2019].

[89] N. Sullivan, “ECDSA: The digital signature algorithm of a better internet,” Cloudflare, 10 March 2014. [Online]. Available: https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/. [Accessed 17 April 2019].

[90] Coindesk, “How Will Ethereum Scale?,” [Online]. Available: https://www.coindesk.com/information/will-ethereum-scale. [Accessed 26 April 2019].

[91] A. Norry, “The History of the Mt Gox Hack: Bitcoin’s Biggest Heist,” Blockonomi, 19 November 2018. [Online]. Available: https://blockonomi.com/mt-gox-hack/. [Accessed 26 April 2019].

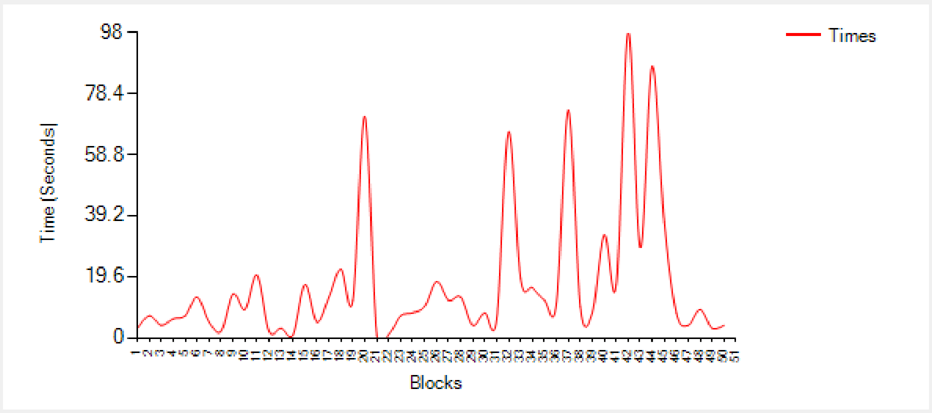
Appendices

Appendix A

Extended Results:

Test 1 (Low transaction rate: 5 TPM):

TestCoin Results:



Max: 98 seconds

Min: 1 seconds

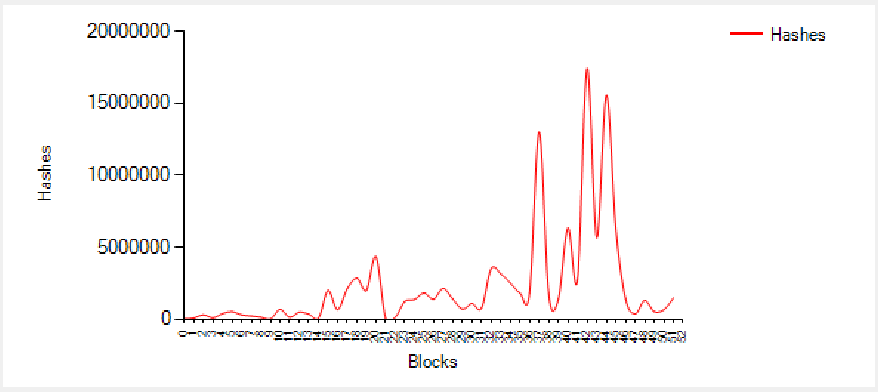
Mean: 17.36 seconds

Range: 97 seconds

Total Time: 868 seconds

Total Blocks 50

Total Trans: 117



Max Hashes: 17409115

Min Hashes: 1836

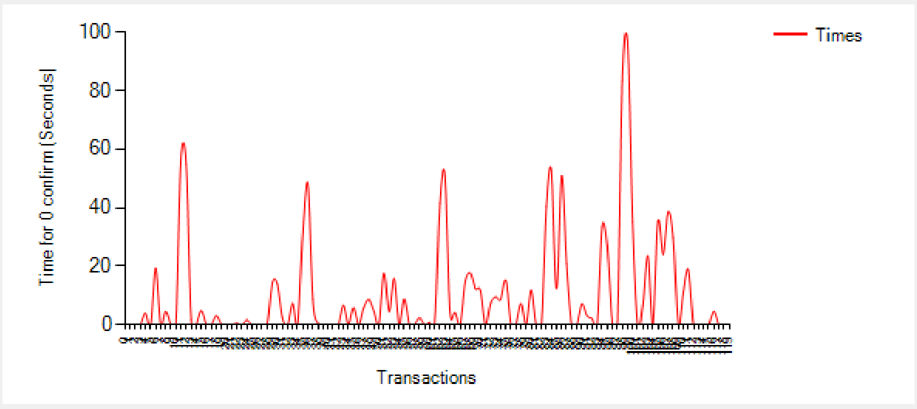
Mean Hashes/Block: 2279436

Mean Hashes/Tran: 1021816.1

Mean Hashes/MaxTrans : 151962.4

Total Blocks: 52

Total Trans: 116



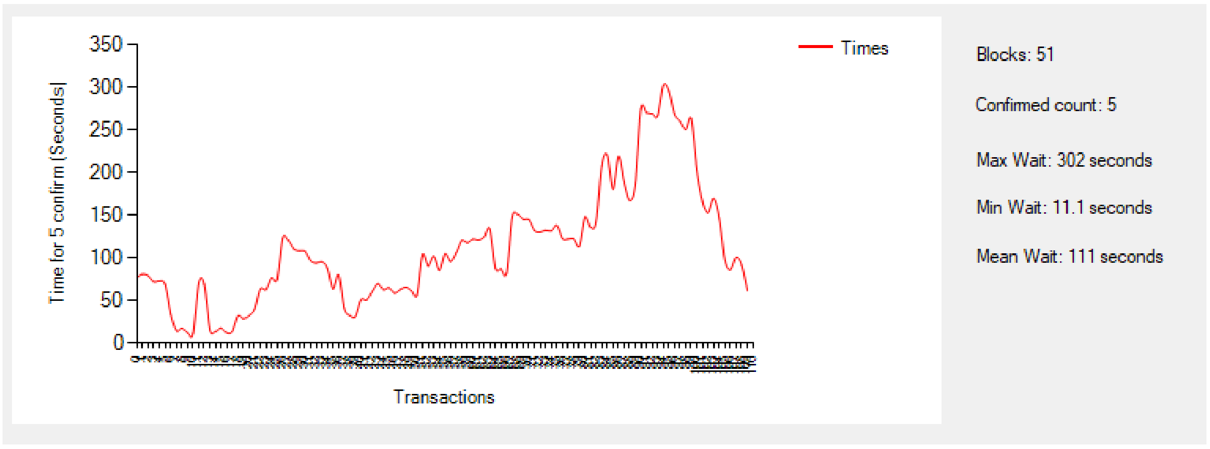
Blocks: 51

Confirmed count: 0

Max Wait: 96 seconds

Min Wait: 0 seconds

Mean Wait: 10.3 seconds



Blocks: 51

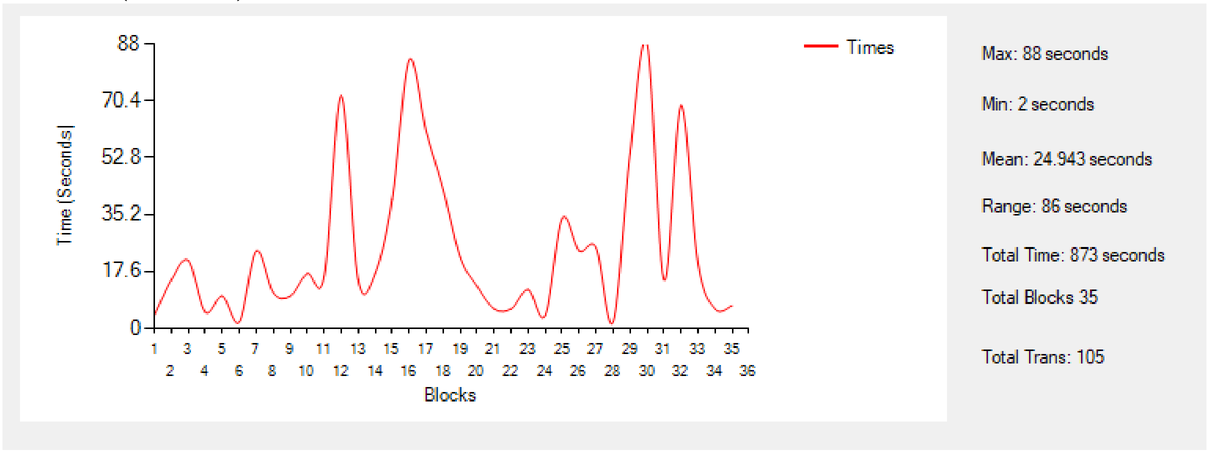
Confirmed count: 5

Max Wait: 302 seconds

Min Wait: 11.1 seconds

Mean Wait: 111 seconds

TestCoin (Throttled) Results:



Max: 88 seconds

Min: 2 seconds

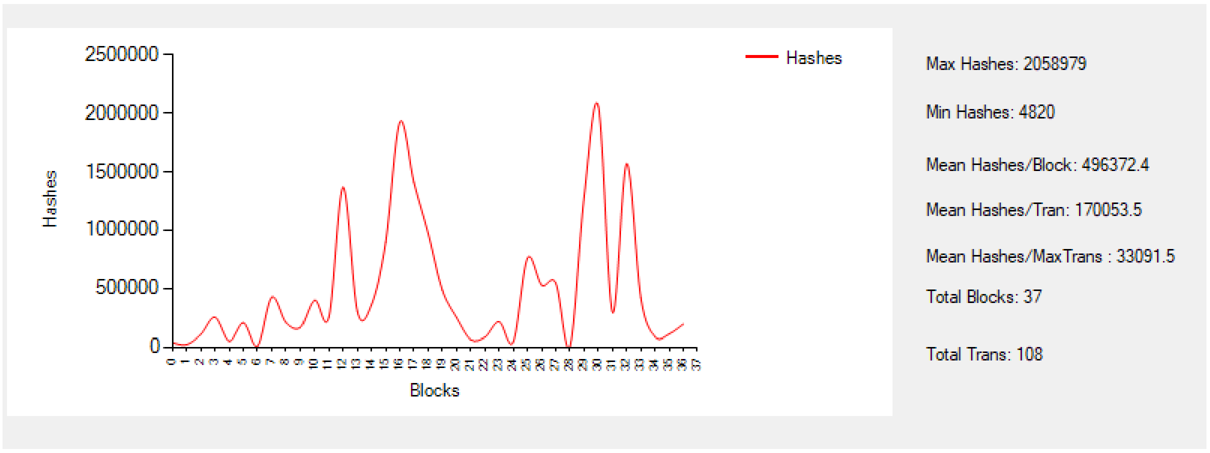
Mean: 24.943 seconds

Range: 86 seconds

Total Time: 873 seconds

Total Blocks 35

Total Trans: 105



Max Hashes: 2058979

Min Hashes: 4820

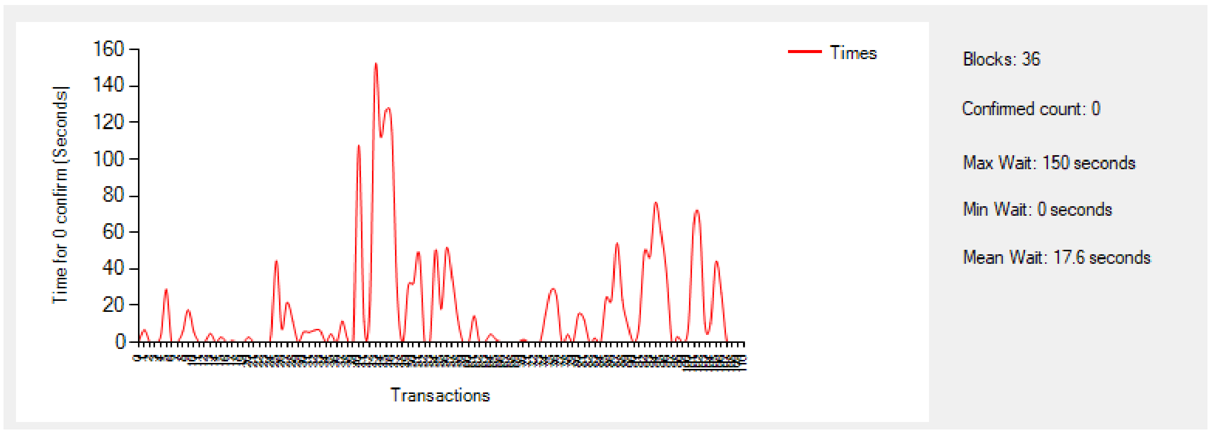
Mean Hashes/Block: 496372.4

Mean Hashes/Tran: 170053.5

Mean Hashes/MaxTrans : 33091.5

Total Blocks: 37

Total Trans: 108



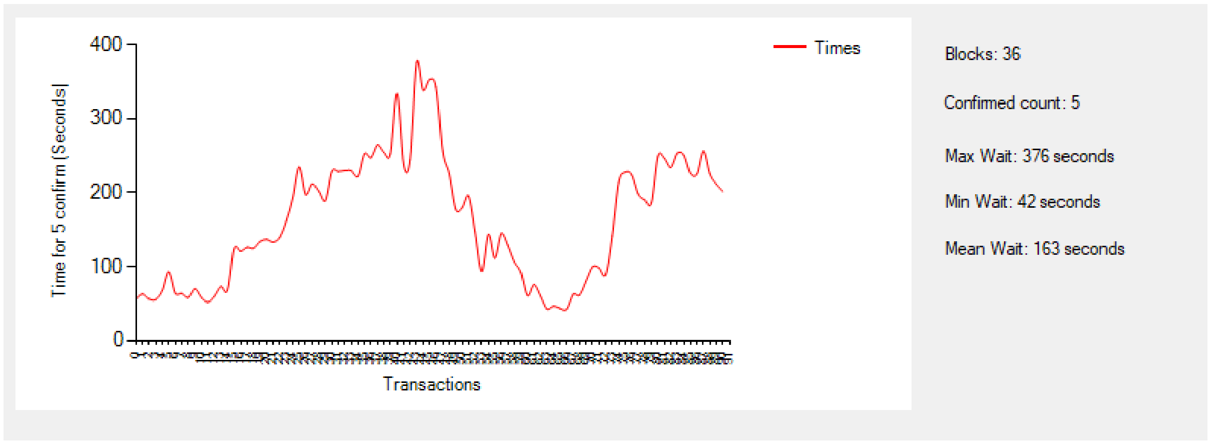
Blocks: 36

Confirmed count: 0

Max Wait: 150 seconds

Min Wait: 0 seconds

Mean Wait: 17.6 seconds



Blocks: 36

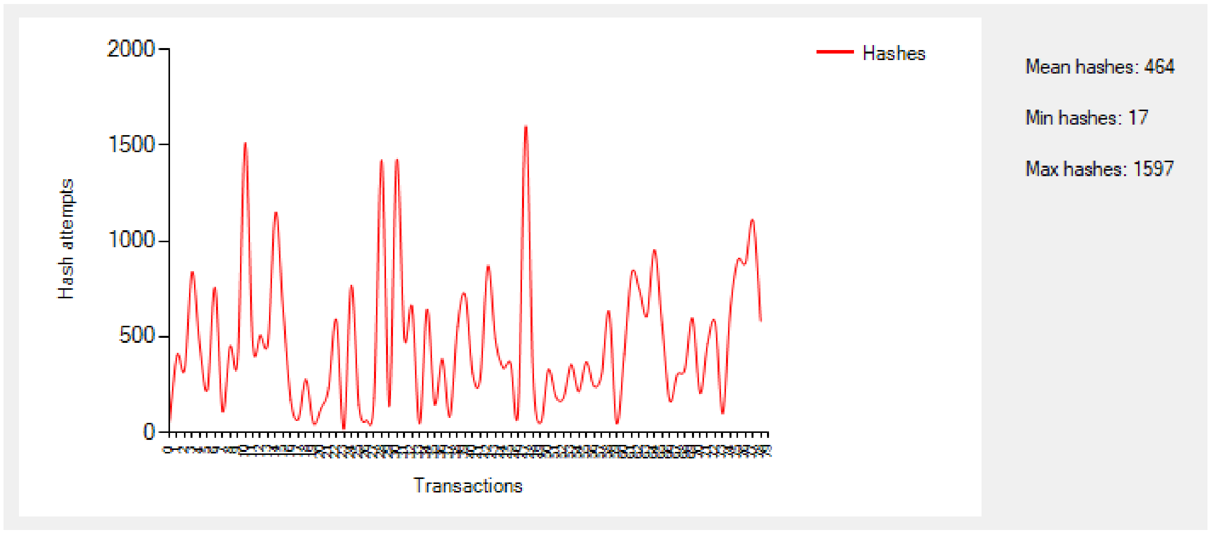
Confirmed count: 5

Max Wait: 376 seconds

Min Wait: 42 seconds

Mean Wait: 163 seconds

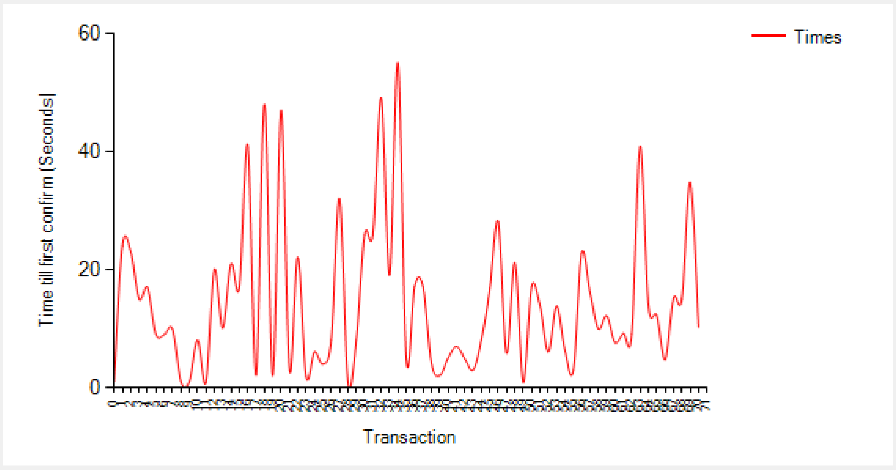
TangleToken Results:



Mean hashes: 464

Min hashes: 17

Max hashes: 1597



Total Transactions: 79

Unconfirmed Trans: 8 10.1%

Mean unconfirmed wait: 456

1+ Confirms: 71 89.9%

2+ Confirms: 41 51.9%

3+ Confirms: 37 46.8%

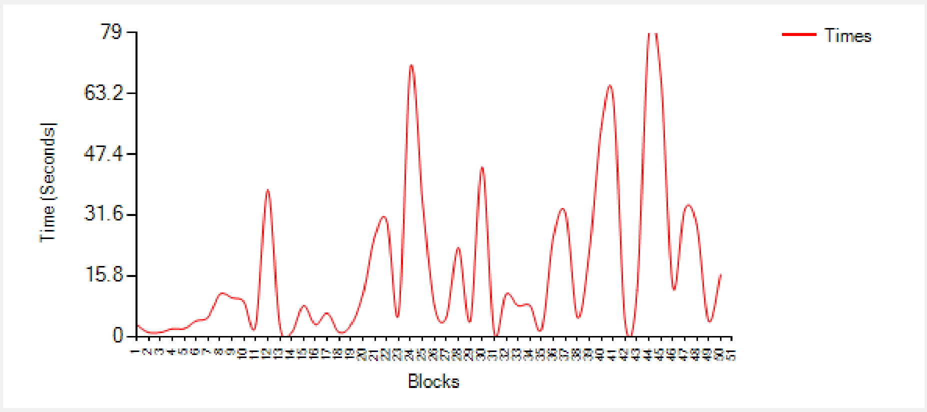
Mean wait time: 14.5

Min wait: 0.88

Max wait: 55

Test 2 (Medium transaction rate: 15 TPM):

TestCoin Results:



Max: 79 seconds

Min: 1 seconds

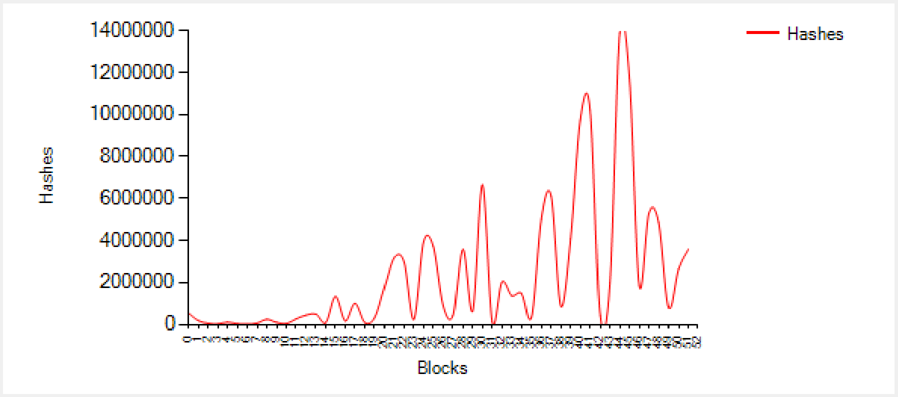
Mean: 17.28 seconds

Range: 78 seconds

Total Time: 864 seconds

Total Blocks 50

Total Trans: 246



Max Hashes: 13868317

Min Hashes: 2419

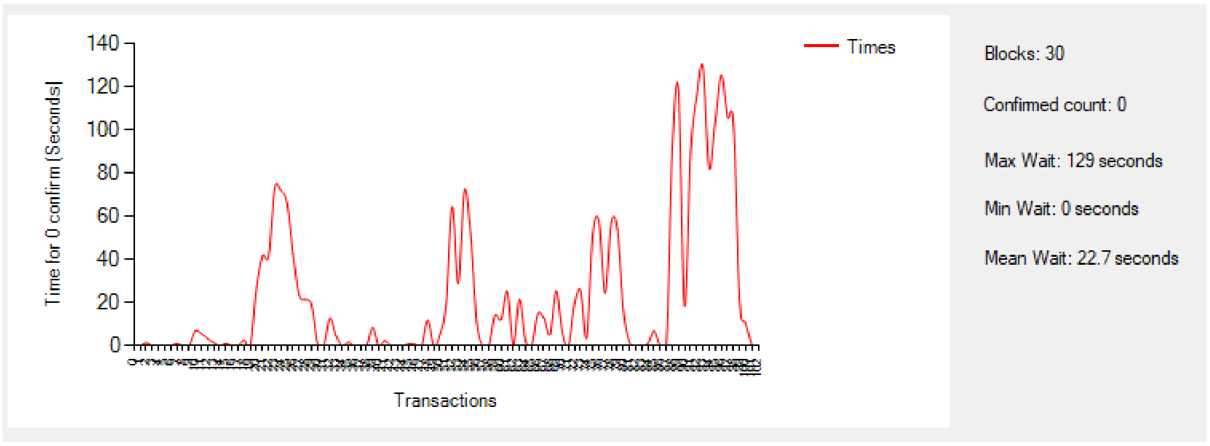
Mean Hashes/Block: 2342519.6

Mean Hashes/Tran: 548698.3

Mean Hashes/MaxTrans : 156168

Total Blocks: 52

Total Trans: 222



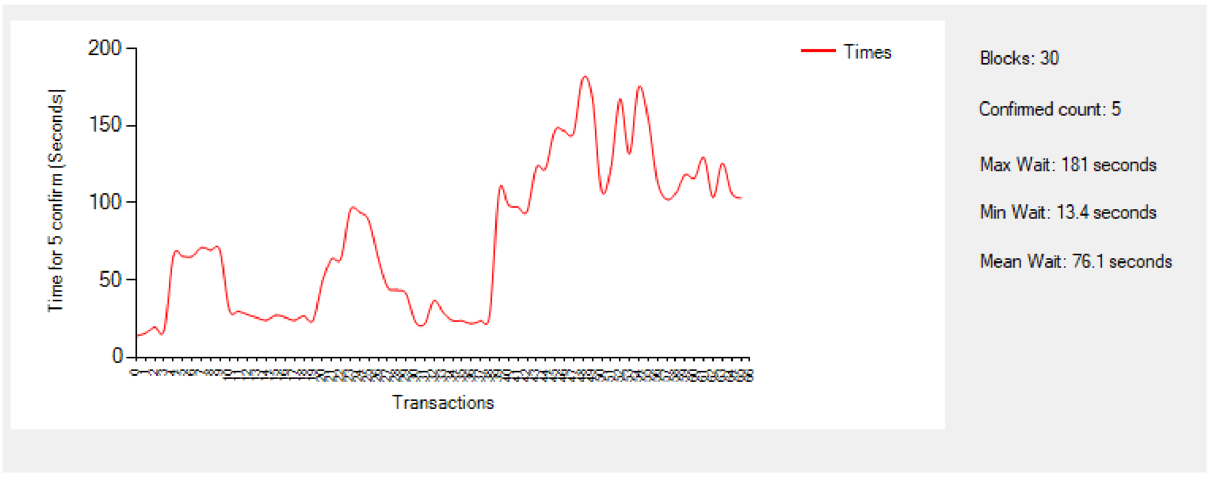
Blocks: 30

Confirmed count: 0

Max Wait: 129 seconds

Min Wait: 0 seconds

Mean Wait: 22.7 seconds



Blocks: 30

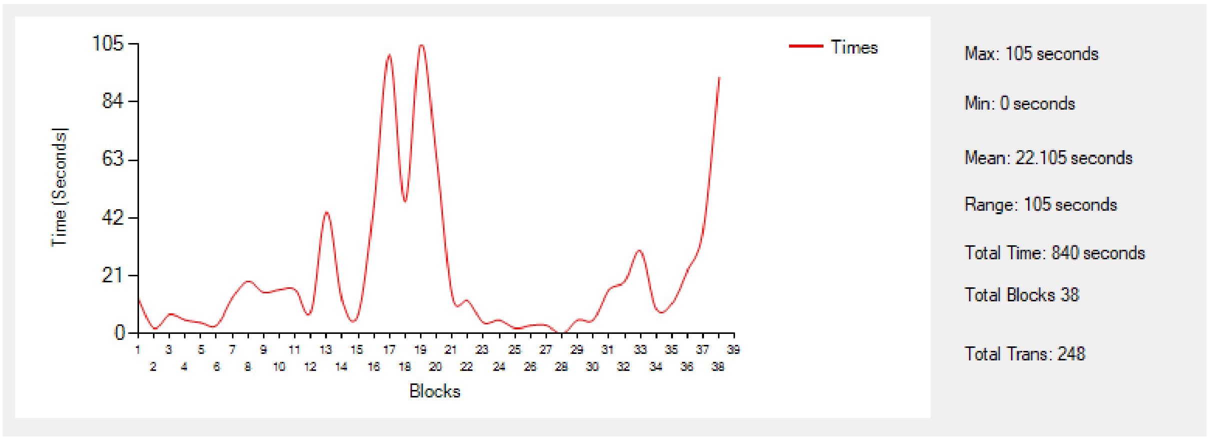
Confirmed count: 5

Max Wait: 181 seconds

Min Wait: 13.4 seconds

Mean Wait: 76.1 seconds

TestCoin (Throttled) Results:



Max: 105 seconds

Min: 0 seconds

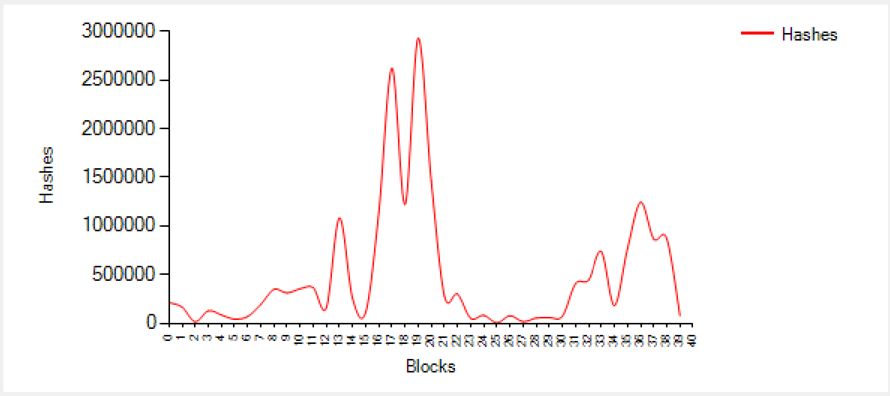
Mean: 22.105 seconds

Range: 105 seconds

Total Time: 840 seconds

Total Blocks 38

Total Trans: 248



Max Hashes: 2928426

Min Hashes: 2025

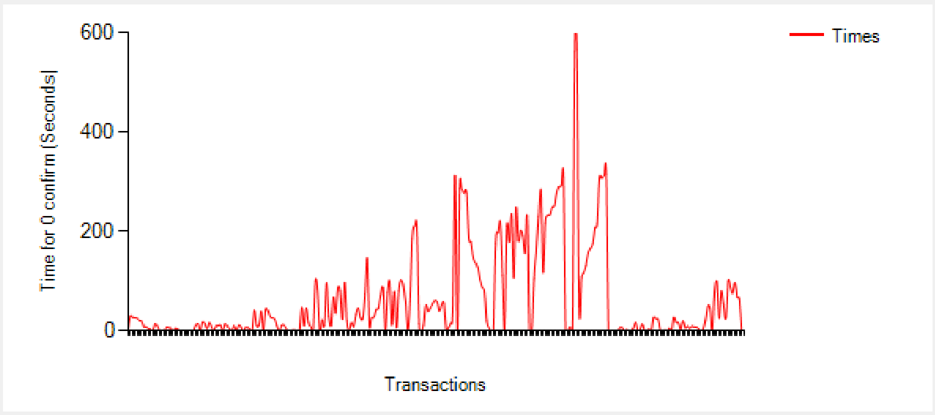
Mean Hashes/Block: 495263

Mean Hashes/Tran: 73645.1

Mean Hashes/MaxTrans : 33017.5

Total Blocks: 40

Total Trans: 269



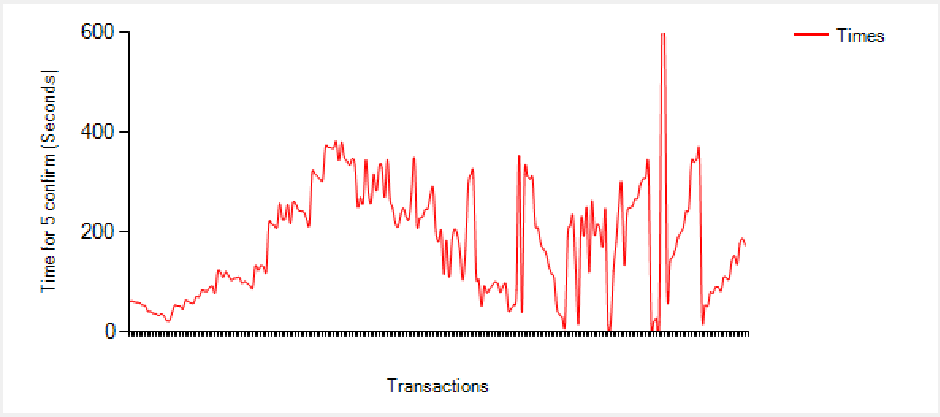
Blocks: 39

Confirmed count: 0

Max Wait: 567 seconds

Min Wait: 0 seconds

Mean Wait: 63.8 seconds



Blocks: 39

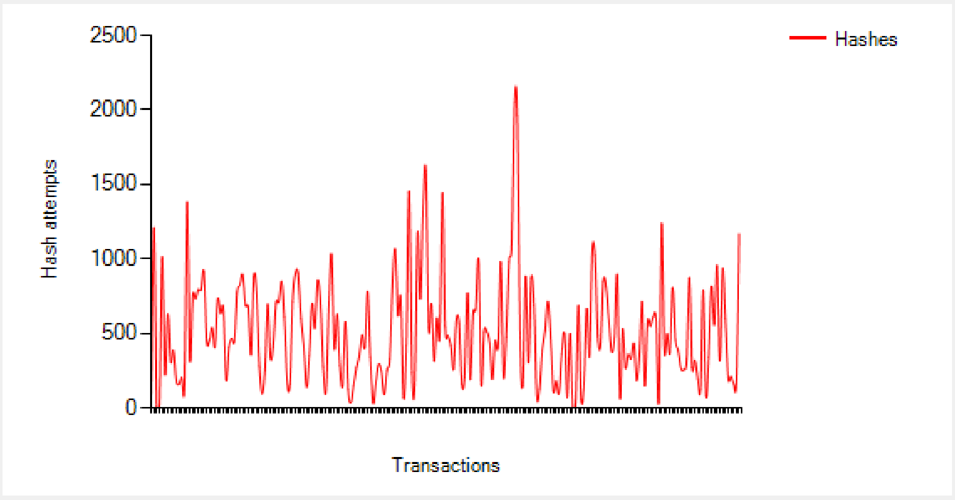
Confirmed count: 5

Max Wait: 600 seconds

Min Wait: 13.5 seconds

Mean Wait: 178 seconds

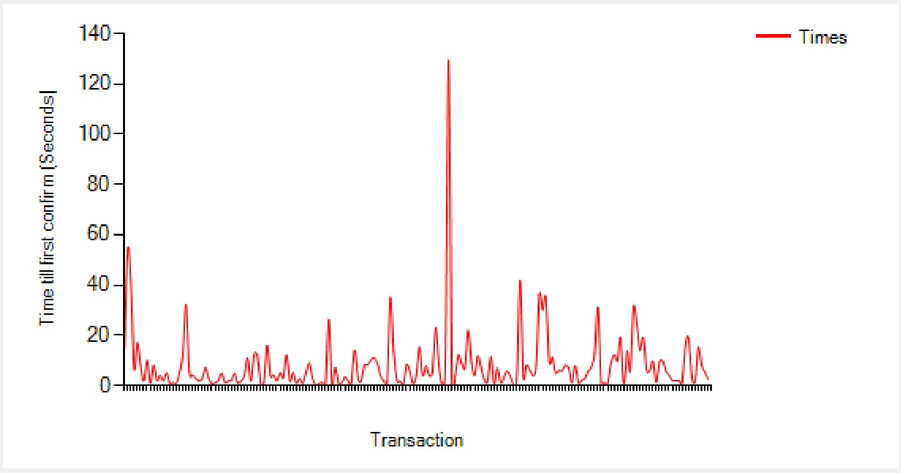
TangleToken Results:



Mean hashes: 515

Min hashes: 10

Max hashes: 2050



Total Transactions: 213

Unconfirmed Trans: 32 15%

Mean unconfirmed wait: 459

1+ Confirms: 181 85%

2+ Confirms: 132 62%

3+ Confirms: 100 46.9%

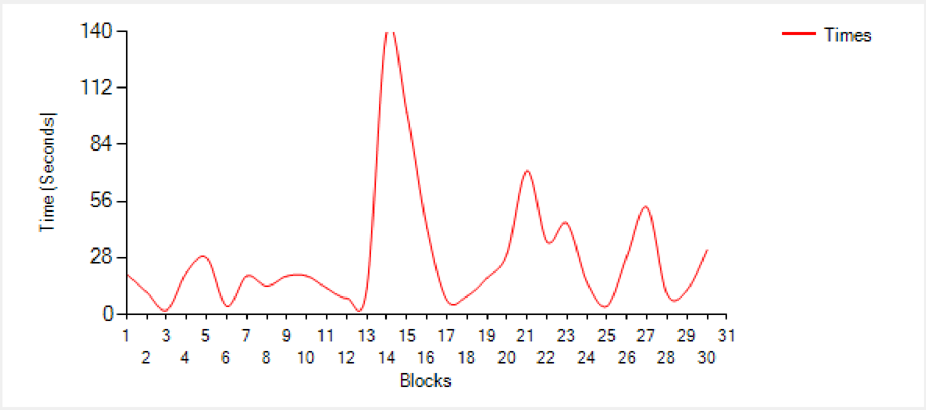
Mean wait time: 8.39

Min wait: 0.24

Max wait: 129

Test 3 (High transaction rate: 60 TPM):

TestCoin (Throttled) Results:



Max: 140 seconds

Min: 2 seconds

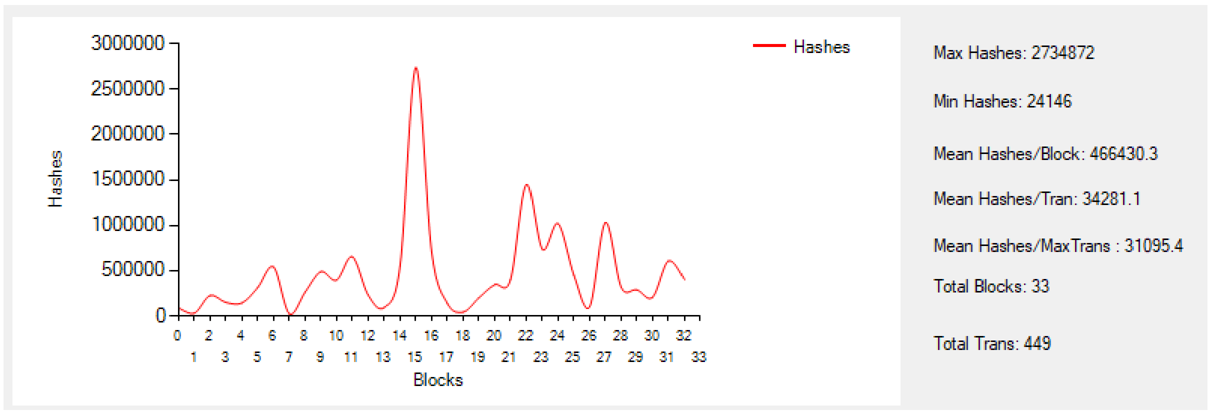
Mean: 28.2 seconds

Range: 138 seconds

Total Time: 846 seconds

Total Blocks 30

Total Trans: 828



Max Hashes: 2734872

Min Hashes: 24146

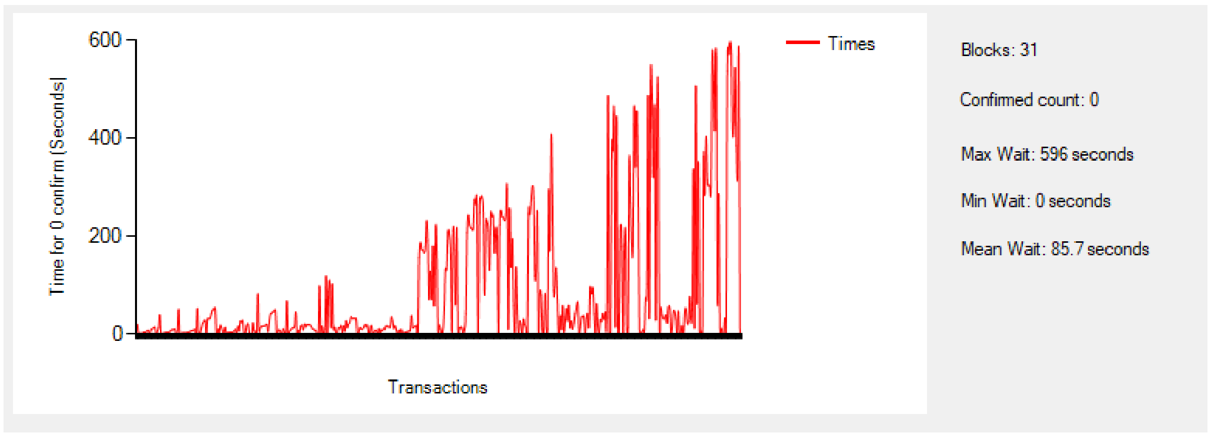
Mean Hashes/Block: 466430.3

Mean Hashes/Tran: 34281.1

Mean Hashes/MaxTrans : 31095.4

Total Blocks: 33

Total Trans: 449



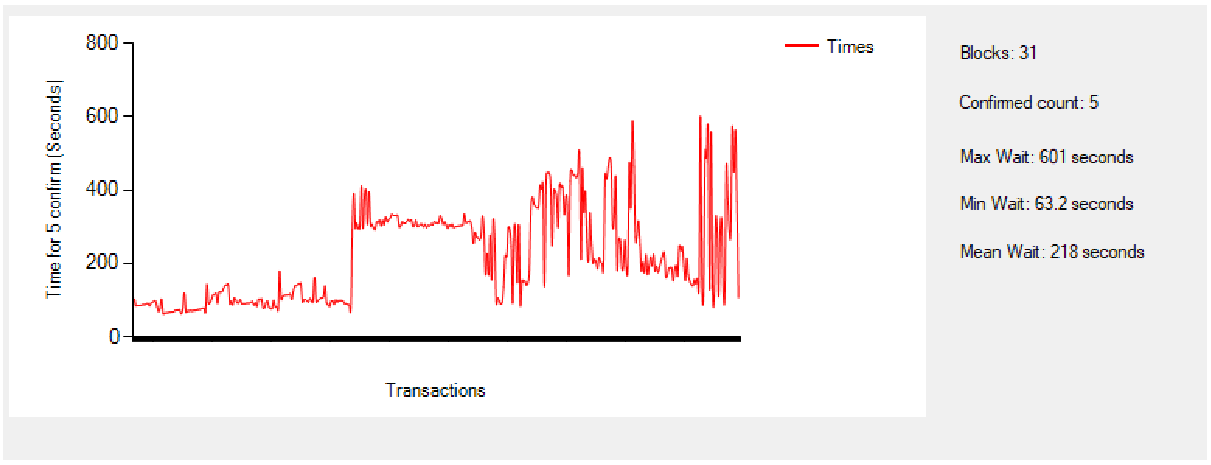
Blocks: 31

Confirmed count: 0

Max Wait: 596 seconds

Min Wait: 0 seconds

Mean Wait: 85.7 seconds



Blocks: 31

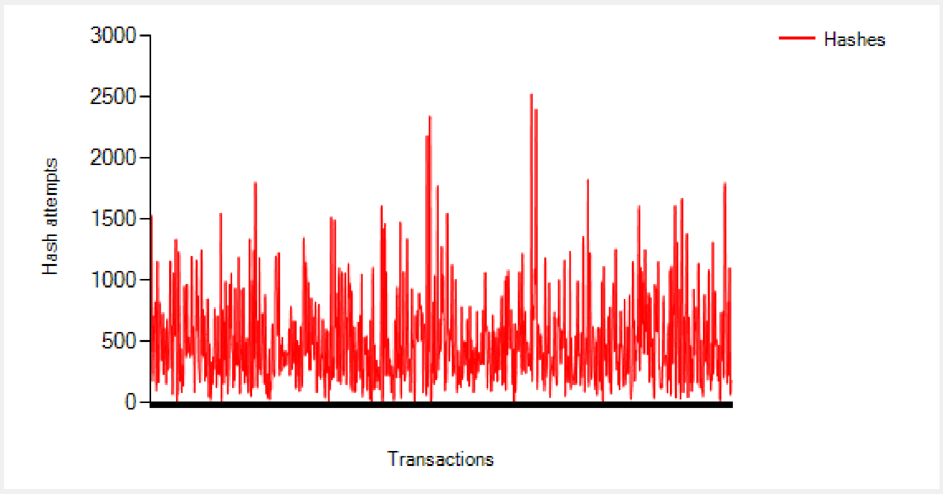
Confirmed count: 5

Max Wait: 601 seconds

Min Wait: 63.2 seconds

Mean Wait: 218 seconds

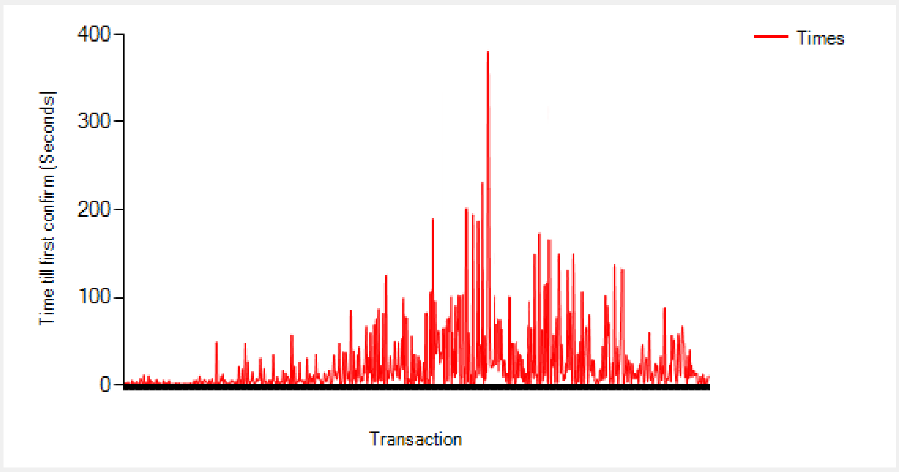
TangleToken Results:



Mean hashes: 504

Min hashes: 3

Max hashes: 2520



Total Transactions: 823

Unconfirmed Trans: 170 20.7%

Mean unconfirmed wait: 426

1+ Confirms: 653 79.3%

2+ Confirms: 429 52.1%

3+ Confirms: 307 37.3%

Mean wait time: 5.67

Min wait: 0.24

Max wait: 376