2017

Seed: A Decentralized Server Ecosystem Using Blockchain Technology

CARSON ROSCOE

COMP 8045

9 CREDITS

PRACTICUM PROPOSAL

Table of Contents

Personal Profile

Computer science has been a hobby of mine, specifically with games development, since I was twelve years old. I began creating videogames in Game Maker originally, and eventually upgraded to Lua, Java, and C++. I became a large fan of cryptocurrencies in high school, spending thousands of dollars to create a mining rig to support the network and earn Litecoins and Dogecoins. I began creating videogames in payment of cryptocurrencies for users on the social media platform Reddit. When I graduated high school, I set aside these hobbies temporarily while I moved away from Vancouver Island to attend BCIT.

Education

Previous, I completed BCIT's Computer System Technology (CST) diploma, graduating from the Data Communications and Internetworking option with distinction. During this time, I completed many successful projects, including ones which resulted in winning an award for Best Personal Project at the BCIT 2016 Open House.

Currently, I am enrolled in BCIT's Bachelor of Technology in Computer Systems, finishing my degree in the Games Development option. I have recently been awarded the Allen and Linda Stefanson Memorial Award due to my academic performance, and will be graduating with distinction.

Work Experience

After completing first year of CST, I landed myself a job as a C# .NET developer at JRP LTD. During this summer, I created Android and iOS applications in Xamarin, as well as learned constraint programming in C++ to write optimizers for our company's existing software.

I proceeded to be rehired and work two more summers at JRP LTD, taking on larger tasks each year. I continued to work primarily in C# and C++ at JRP, however I was moved away from mobile app development and moved primarily onto Windows application development during the second summer.

In the third summer, I was personally tasked with creating the proof of concept for our software that was then submitted as our application in a bidding war for a multimillion dollar contract. A key feature in this project was creating a decentralized peer-to-peer caching system across multiple computers on the network.

Area of Specialization

My specializations range primarily from data communications to games development. I chose the Data Communications diploma option for CST, as well as the Games Development option for BTECH, both specifically to aide me in following my dreams of creating networked games. Over the last few years, my interests have also began including blockchain technologies more and more.

I previous was a cryptocurrency miner for Litecoin and Dogecoin, created my first Smart Contract for the NEO cryptocurrency ecosystem during this past summer, and began prototyping my very own blockchain in Python. My recent desire has been to merge all my specializations, pairing the security and structure of decentralized blockchain technology with networked videogames.

Description

This project aims to explore using blockchain technology to create a provably-fair decentralized videogame server. A blockchain protocol will be designed and implemented for this purpose. The protocol will be implemented in web technologies to prove its effectiveness in it's desired use case. The protocol and implementation will replace the traditional competitive proof-of-work mechanism found in cryptocurrencies with a new mechanism that will be referred to as proof-of-gameplay. This project will also be designed around being modular, where it can be reused for multiple games.

Due to the nature of this projects cryptography and networking based requirements, I would like to request Aman Abdulla be my designated supervisor.

Background

The goal of this project is to create a decentralized system that allows for an online multiplayer game to exist without a centralized server, and accomplishing this by utilizing blockchain technology. Decentralized game servers have been a challenge in the past, with one of the biggest flaws being the lack of authority to dictate the truth when a user decides to cheat in peer-to-peer play. If a mechanism was implemented that allowed for complete trust of the information being passed into the game, decentralized servers would have the potential to be much more common place. Blockchains are a rapidly growing technology that allows for decentralized trustless communication to occur between multiple parties without a centralized authority. This project plans on using blockchains to solve the issues with trust that occur in decentralized servers. This technology also allows for the game to be provably fair, with all logic and communication being open for all clients to judge and validate each other. Traditionally, blockchains were used for trustless systems that desired security over all else, such as with the most famous example, Bitcoin. Bitcoin was the first electronic cash built on blockchain technology, with many alternatives following all known as cryptocurrencies.

These cryptocurrencies were created with different constraints than that of a videogame, as they were intended to be used in place of a currency, rather than a live videogame server. One primary difference in constraints is that of speed. In Bitcoin, transactions can be sent out by anyone at any time. A type of user, known as miners, listens for these transactions and adds them to a data structure known as a block. A new block is created and propagated on the network every ten minutes, meaning that when the network is running smoothly, there is roughly a ten-minute turn around time for sending and receiving transactions. At this point, however, the transactions are unconfirmed, and users are supposed to wait another six blocks until we can guarantee with complete certainty that the block is an accurate representation of the transaction history. Bitcoin does not care about speed as much as a game server does. A bank wire transfer can take an entire week, so since Bitcoin is attempting at emulating a currency, speed is not a high priority. Online videogames can have varying network requirements, some of which needing to send messages every few seconds, and others needing to send multiple per second. Therefore, this project must modify the underlying validation and block creation mechanism to speed up the system.

The largest bottleneck in Bitcoin is that only one megabyte of transactions is stored in a block, which is created every ten minutes, meaning only one hundred kilobytes worth of data can be transacted a minute on the entire network. Although the size of a block and creation interval are arbitrarily defined, this bottleneck exists due to the proof-of-work mechanism used for validating the computing usage

exhausted by miners during block creation. Alternatives to proof-of-work exist, such as proof-of-stake, proof-of-activity, ghost and proof-of-burn. Each of these alternatives solved a separate issue with proof-of-work, however each come with their own flaws as well.

This project will require a new mechanism to be developed for it, which will be referred to as proof-of-gameplay. This mechanism will allow for the replacement of competitive miners in the network, instead replacing them with a validation approach similar to that of the Tangle used in the IOTA cryptocurrency. Each transaction will include validation proof of previous transactions, allowing for transactions to be trusted without waiting for miners to mine blocks.

This project will also require a reconstruction of the blockchain data structure. Blockchains are traditionally a chain of blocks, effectively a linked list of block objects, who each know of the block before them. The size of these blockchains will increase with each new block, raising the issue of memory storage. In a cryptocurrency, we need to store all previous information as we care about the history and accuracy of all transactions. However, in a live server, not all transactions need to be stored in long term storage. For example, if we joined a game and a goblin had been killed beside where we were standing a minute prior, we would care about that, as we most likely need to render the corps. We do not, however, need to know every goblin who has ever been killed, and by who. This difference in requirements leads to a difference in design decisions, namely what information is stored on the blockchain. The Seed System redesigns how blockchains function, mixing concepts such as IOTA's tangle, transaction squashing and circular blockchains to maximize storage efficiency while minimizing transaction confirmation wait times.

Innovation

The seed system is a blockchain protocol to allow use of a decentralized blockchain server for multiple games. Blockchain technology is a very new field, with many variations, both tested and non-tested, surfacing every month. Cryptocurrencies such as Ethereum or NEO have risen which allowed for code to be executed on the blockchain. Cryptocurrencies such as IOTA have restructured blockchains and the ledger, allowing for an alternative to block mining to be required for transaction confirmations. Unique ideas have surfaced in isolation, however no system has taken from these concepts and combined them in such a way that allowed for a decentralized live game server.

The first source of innovation is found in the seed systems design. The system has two separate chains that branch out of the genesis block, one for temporary storage, while the other is for permanent storage. The temporary storage uses a circular buffer styled blockchain of fixed length, with dynamically sized blocks, to store temporary data. The permanent storage side uses an entanglement system like that of IOTA, to receive transactions out of order, and then stores these entangled transactions into blocks to squash them then add to the permanent blockchain. This design allows for a separation of importance between transaction types, effectively giving the system an ordered quick temporary storage blockchain, or an unordered delayed permanent storage blockchain.

Another source of innovation in the system is the concept of transaction squashing. When the entangled transactions are put into blocks, to reduce space, these transactions become squashed. A squashed transaction is a simplified version of history, where the difference between the start and end states is recorded, not the steps to achieve the end state. For example, if there were three transactions, and in each one User A sent 10 units of currency to User B, the squashed recorded history would be a single

update which stated User A sent 30 units of currency to User B. This concept will simplify recorded history, drastically reduce the storage size, as well as keep the integrity of the system in tact.

The modular reuse of the system between multiple games is another source of innovation. Instead of having hardcoded game logic in the blockchain, the system will interpret commands based on the defined module a transaction claims it belongs too. This adds a level of complexity to the users of the system, as they must define the specifics of their own game logic, however it adds a level of modularity to the system that is unprecedent.

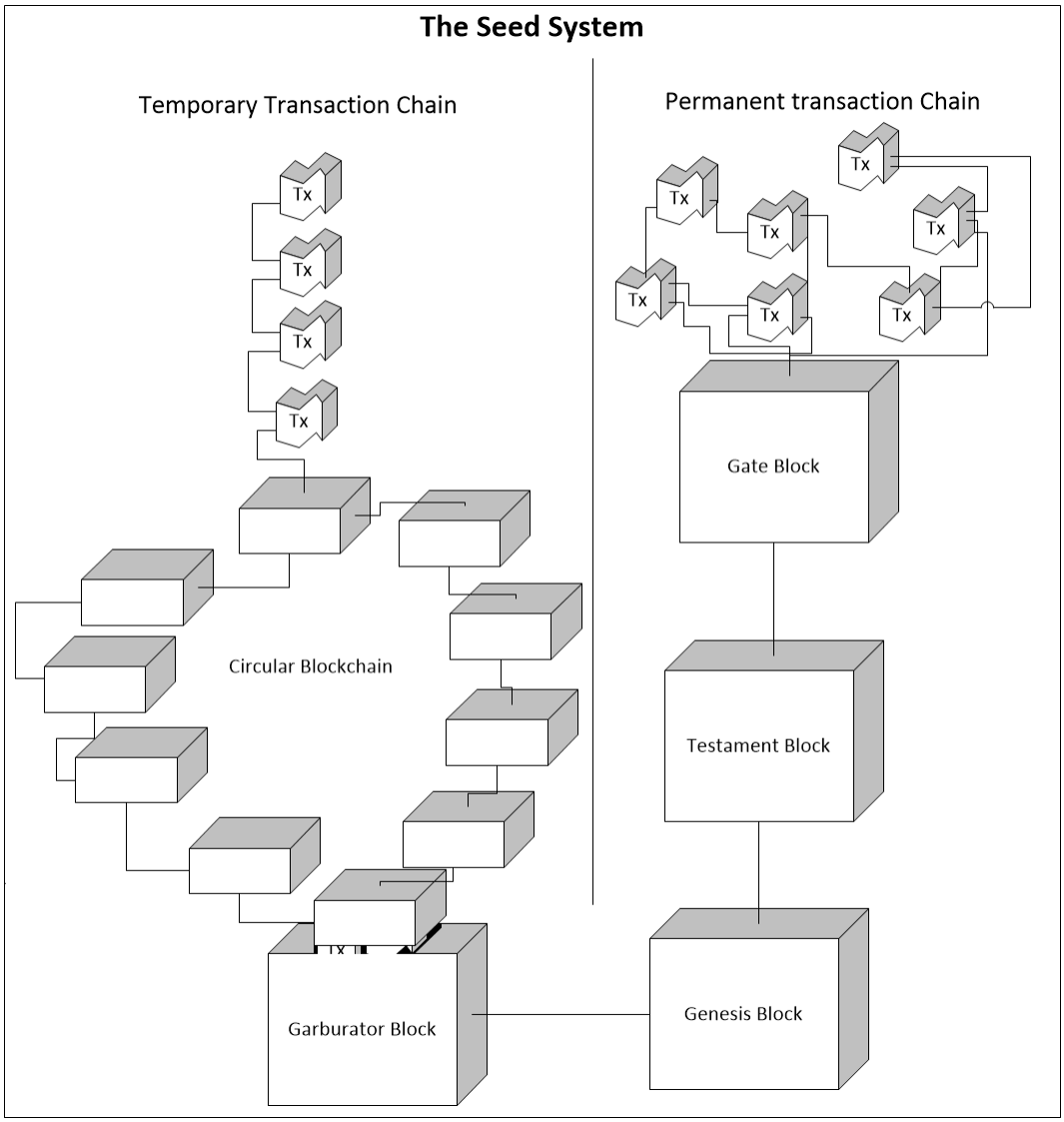
Different modules, as well, will be able to nominate trusted nodes in which they trust. This would mean that the blockchains run fully decentralized, however trust of a specific module can be given to certain nodes, which effectively get a large voting right in the network. This would allow for modules themselves to choose to forgo pure decentralization in the name of network speed, without harming the legitimacy of the rest of the network. An example could be that Game A nominates ten users, which are their own servers, to act as trusted nodes. Users playing their game would follow the regular decentralized rules with each other, however if they receive a transaction being propagated from a trusted node, they would instantly trust that transaction as they trust it's source inherently. This would reduce propagation time on the network for modules that choose to implement it.

The final innovation that is to arise is the cryptocurrency inside the blockchain known as Seed. Seed is the currency of the network, which will aim to have real world value and use like most cryptocurrencies. Seed is not a in-game currency that modules share, instead it is a different module altogether that acts as the default module all addresses use upon creation. Essentially, Seed uses the system just like any other module, however it is used exactly like a traditional cryptocurrency, with limited extra functionality. Commands do not exist across modules; therefore, seed cannot be used to buy in-game items of any modules, nor can it be used to trade value from one module to another. It is simply its own cryptocurrency, which can be traded between any users on the network. The seed system, modules and coins are outlined in more detail in the following Seed section.

Seed

Seed is both the name of the entire ecosystem of the seed system, as well as the name of the cryptocurrency of the seed system. For the purposes of this proposal, the currency token will be referred to as Seed token, while the ecosystem is referred to as the Seed system.

Seed System Design



The seed system is a combination of subsystems that click together to create a decorated blockchain that can handle both rapid succession temporary transactions, as well as unordered permanent transactions.

Genesis Block

The genesis block is the first block in the seed system. It contains the hardcoded logic for the initial state of the system, as well as the deployment code for the first module on the system which represents the

cryptocurrency of the seed ecosystem. Both the temporary transaction chain and the permanent transaction chain branch off the genesis block.

Permanent Transaction Chain

The seed subsystem which handles all permanent transactions. Incoming transactions are entangled into the upper entanglement. Entangled transactions are eventually moved into the Gate Block, where they are ordered, fully confirmed, squashed and then moved into the Testament Block, which is the first block after the Genesis Block.

Entanglement

The entanglement is the grouping of tangled permanent transactions. When a transaction is received, it contains validation work on previous transactions in the tangle. These transaction validations may either be for transactions currently in the upper entanglement, or ones that are being held in the Gate Block, which holds the lower entanglement. Once a transaction is added to the entanglement, it is unconfirmed, however it being added helped confirm other previous transactions, pushing the system along forward. After more transactions are received, incoming transactions will validate the original transaction, allowing more faith to be put in the validity of the received transaction. Once the entanglement reaches a certain size, the system pushes the upper entanglement transactions that are confirmed enough to be trusted into the Gate Block.

Gate Block

The Gate Block is a block which holds the lower entanglement inside it. These transactions are considered confirmed. When a new grouping of entangled transactions is added, the transactions currently held in the Gate Block are squashed and then added to the Testament Block.

Testament Block

The testament block is the first block after the Genesis Block. It holds the current squashed transactions of every permanent transaction on the network to date. When the Gate Block adds a list of squashed transactions to the Testament Block, they are merged together into the testament block. The testament block acts as one large squash of transactions, allowing it to take up very little disk space. For example, if the following scenario of transactions was received from the Gate Block:

Sender	Receiver	Amount
User C	User B	100
User B	User A	50
User B	User A	50
User B	User A	50
User A	User B	25

The testament block would store a squashed version of the transaction info, which effectively is stored as the following.

User	Amount Changed
User C	-100
User B	+25
User A	+125

This compression of data is essential for the scalability of the network. When a user appears on the network, they can simply request the current Testament Block, apply those changes to the Genesis Block, and see the current state of the confirmed world.

Temporary Transaction Chain

The seed subsystem which handles all temporary transactions. Incoming transactions are added to the circular blockchain and eventually passed over to the Garburator block, which is the first block after the Genesis Block.

Circular Blockchain

The circular blockchain is a blockchain that, instead of being structured like a linked list, instead of structured like a circular buffer where block storage is reused after a while. This system stores the temporary transactions, with each transaction added to the block containing validation information on previous temporary blocks, proving that this transaction both validates a previous block in history as their contribution, while also confirming they are in fact in agreement with the chain they are attempting to join. If a transaction is received which disagrees with an existing block on the circular blockchain, it will state which transaction in the block it deemed invalid. Transactions which were deemed invalid by the majority of incoming transactions are considered invalid and discarded from the blockchain. This blockchain will run very fast, attempting to achieve two blocks a second speed with dynamically sized amounts of transactions per block. Once a block is about to be overwritten with new transactions, its existing transactions are passed over to the Garburator Block, which simply contains the hashes of all discarded transactions. The block is then overwritten with new contents, which are new unconfirmed transactions.

Garburator Block

The ever-growing block on the temporary blockchain. When the circular blockchain discards confirmed transactions, it disposes of them into the garburator block. This block strips all the unnecessary contents out of the blocks, only keeping with it transaction hashes. It then stores the checksum of these transaction hashes with the checksum of their blocks hash. This allows transactions to reference previous transactions and have the system confirm that a transaction did occur with a given hash and it was validated, however it's contents are irretrievable as it's been disposed.

Modules

A module is defined server logic that dictates the rules for a given modules implementation. Transactions for a given module must be validated according to that modules rules. Modules are, essentially, an abstraction layer of server logic, allowing for multiple different modules to run on the same system, validating transactions from other modules. Instructions for transaction validation are given in a modules implementation by in comma delimited commands. Modules should be able to do complicated checks such as confirm a user has the correct units of a given type before they can purchase another, or confirm that a user's position data is within a certain range of another user's position data before attacking.

Modules will allow for key value pair data storage of undefined structure. For example, a module may define that each user has a structure of data which represents an inventory, as well as a structure of data which represents their level progress. Modules can define the structure of their data, as well as set the rules for what can and cannot be modified.

When a transaction is to be created, it must validate previous transactions to contribute work to the network. These transaction verifications are module agnostic. A transaction for Module A can validate transactions from Module B as their contribution to the network, and vice versa. Designing the system this way allows for each module to scale with the next, as well as allow for the upkeep of a modules network while user is actively using such module. For example, a user can transact for Module A on one day, and another user can transact for Module A on another day, and those transactions would still go through despite no other active users using the network during those times, simply because Module B's community validated the transactions for Module A.

Seed Currency Module

The Seed Currency Module is the initial, default module deployed to the system, which is hardcoded into the genesis block. This module defines a single variable, Seed, which is to be used as the digital currency of the network. This module will define rules for transactions with Seed, such as requiring users to own at least the amount they intend to transfer, as well as require all Seed transactions be on the permanent transaction chain rather than the temporary transaction chain.

This modules currency is the currency that powers the network. It is connected to the proof-of-gameplay mechanism in the currency. Proof-of-gameplay takes elements from proof-of-work, proof-of-stake and proof-of-activity to create an alternative mechanism which aligns the desires of all users of the network, rewards users for actively using the network and rewards users for validating fellow users' transactions.

Users pay on the network for every transaction they submit, while also being paid in seed on the network by validating other user's transactions when they submit a transaction. Effectively, if a user validates three other transactions every time they transact, they will pay zero fees to the network as their support of the network is paid off by cooperatively validating a fellow user's transactions. If users choose, they can refuse to validate and pay the network in Seed instead. If users want to earn Seed, they may choose to validate more than three transactions per outgoing transaction, creating a surplus of validation. This validation adds extra security to the network, and reduces validation propagation time as more sources around the globe are validating any transactions.

Based on a set interval, Seed will be periodically released to users on the network. The breakdown of earned seed will be based on each user's activity and their validated transaction work. The amount of transactions a user submitted since the last payout, added to the amount of transactions they validated since the last payout, is added together to represent their stake in the pool. When that pool of money is released, it is split to all the users based on their stake.

For example, suppose a pool of 1000 Seed were released every hour, and the following table represents the amount of transactions and validations each user did during that hour. The final column represents how much Seed would be paid out to each user.

User	Transactions Sent	Transactions Validated	Stake in Pool	% Of Pool	Seed Payout
A	10	30	40	5.37%	53.69
B	10	60	70	9.40%	93.96
C	5	30	35	4.70%	46.98
D	100	300	400	53.69%	536.91
E	20	0	20	2.68%	26.85
F	25	75	100	13.42%	134.23
G	20	60	80	10.74%	107.38
Total:	**190**	**555**	**745**	**100%**	**1000.00**

This system showcases how the seed payout is a good representation of your network activity and contributions. User D is a hyperactive user who only validates three transactions at a time, however they used the network much more than any other user. They used more and validated more, however they brought the largest amount of validation with them, and were paid out accordingly. User C represents a user who barely used the system, however validated twice as much per transactions than required. We can see that, despite barely using the system, this user received nearly the same amount as User A who used it twice as much, despite validating the same amount of transactions. This importance on transactions sent is there to reward users for using the network, as well as to disincentives pure miners, users who validate much more than they transact in order to gain coins, from taking over the network. User E is an example of a user who did not confirm transactions and instead chose to pay the network instead. In a more detailed example, their Seeds sent would have been added to the 1000 Seed pool, showcasing how existing users are rewarded more when other users chose to pay instead of work.

In proof-of-work systems, the coin distribution is isolated to the top miners, while in a proof-of-stake system, the coin distribution is isolated to the richest users. This proof-of-gameplay system keeps the coins constantly distributed to average users, preventing the wealth from being isolated.

Scope

The majority of the requirements are found in the table below.

Functional Requirements

The Seed System

#	Description
1	Two hardcoded chains will fork out of the Genesis block
2	One fork leads to the Garburator block
3	The Garburator Block can take in transactions, hash them, store the hash then discard it
4	The Circular Blockchain takes in transactions, each of which have validation of another block on the chain
5	The Circular Blockchain removes bad transactions from blocks as they get validated by incoming transactions
6	The Circular Blockchain passes validated transactions to the Garburator when it must overwrite a block
7	The other fork leads to a Testament Block
8	The Testament Block holds a squashed list of transactions merged together
9	The Gate Block takes in transactions from the Lower Entanglement
10	The Gate Block can squash transactions and feed them to the Testament Block whenever it gets a new wave of transactions
11	The Entanglement can take in new transactions, add them to the entanglement, and check their validations to confirm other transactions
12	The Entanglement can pass its deeper entangled lower half into the Gate Block when it reaches a maximum capacity
13	The Seed Systems can parse transactions and determine whether they get added to the Circular Blockchain or the Entanglement based on whether they are a temporary or permanent transaction
14	A ledger can recreate the world by requesting the Testament Block and applying the squashed changes

Cryptographic Requirements

#	Description
1	Users can generate a private key securely
2	Private Keys can be one-way hashed via SHA256 to create the public key
3	Public Keys can be double SHA256 hashed to create a public hash
4	Public Hash's can be Base58Check encoded to create a Public Address
5	Transaction hashes can be signed by a user's private key
6	Signed transactions can have their signage validated by verifying it was signed by a user via their public key

Module Requirements

#	Description
1	Modules can specify key/value pair structures that exist for storage on a per-user basis
2	Modules can specify global key/value pair structures for the entire module
3	Modules can specify verification logic for transaction verification
4	Client's module implementation can specify the execution of transactions
5	Modules can nominate specific public keys to act as trusted nodes for transactions of that module type
6	Modules can read from other modules' data locations; however, they cannot write to them

Network Propagation / Validation Requirements

#	Description
1	Relay nodes act as decentralized user registry's. They keep track of which IP's are online
2	Users can connect to a relay node to be told of users near them (People who are worthy of propagating to)
3	When a user connects, the relay node they connected to tells all the other relay nodes about this user
4	Users can send Transactions to users near them and to relay nodes
5	Relay nodes relay all transactions to other relay nodes and to everyone they know about on the network
6	When a user disconnects from a relay node, it unsubscribes them and tells all other nodes/users near them to not send to that user anymore

Non-Functional Requirements

Flexibility

The module system should be powerful enough that any client running any module can receive transactions from a different module that this client is not developed for, and still be able to validate the transaction to support the system. That means the power of the commands that can be used to build validation functions for a module must be strong enough that it can all be handled without running the actual client for that module.

The module system should be modular enough that it can power a variety of projects, such as a cryptocurrency or a near live multiplayer online game server.

Scalability

The system should scale with more users and modules. The more users, the more confirmations, the quicker the network speed. The system should be able to handle enough users to power an online game without slowing down noticeably.

Technical Challenges

Scope

Scope is an issue in this project. The project is using recent technology, and aiming be strong enough to support an actual online networked game.

Performance

Gaining performance from the system will be a challenge, as confirmation times, as well as propagation time, both are concerns that have not been adequately addressed. We may find that those two factors are too slow for a system as speed reliant as it.

Scalability

The system intends on scaling with an increase in traffic. This approach to blockchain to achieve this is not well tested, and may prove to not work as planned.

Flexibility

The flexibility of the system is a large requirement. It will require a solid design with a lot of planning to do properly in an efficient manner.

Technical Knowledge

The project requires a high degree of technical knowledge in an innovative technology, advancing in a field with minimal academic research.

Methodology

This research project will be conducted following a modified version of agile scrum.

This project is being developed alongside another research project created by Jaegar Sarauer. His research project is to create a 2D JavaScript MMO videogame, in which this project will power the networking. His project will have its server logic written into another module on this system.

Due to the positions of our projects, his project is effectively this projects' first client. Our agile scrum meetings will coexist as we work side by side on two separate projects. This projects' deliverables will be handed off to him as the system is more and more usable.

This project will also be following test driven design. As such, part of the bi-weekly deliverables will include all the tests for each component being delivered.

This project will be following sprint durations of two weeks.

Technologies

The protocol being developed does not require any technologies. It, like any other blockchain protocol, can be implemented in any language and should be able to fully comply with all other clients.

The implementation being developed will initially target web browsers. The purpose of this is to prove that it can be developed for a web browser MMO videogame, as that is the desired use case attempted to be solved with this project.

For this implementation, JavaScript will be used along with Node.js for the required socket connection. A cryptography library capable of doing SHA256 hashing and Base58Check encoding will also be required. Jasmine, a JavaScript library for Unit Testing, will be used to accomplish all testing.

Detailed test plan

This project will be developed following test-driven design. Unit testing will be used to test the expected behaviours of methods as components are developed, as well as test the behaviour and expected outputs of larger systems which are built on the tested components. Every behaviour in the blockchain should be deterministic, making it the perfect candidate for such testing.

Unit Testing

The JavaScript library Jasmine will be used for the creation and execution of unit tests, as it is supported in all major browsers. Components will be developed following test-driven design, requiring that nearly every component should have tests written for it. Ideally, following this design should reduce the number of unforeseen bugs, despite it requiring a little extra boilerplate work. Unit tests will be created every sprint. All unit tests will be run as recent changes come in to confirm that old components still function as required.

Test Case Examples

Transaction

#	Test Description
1	Transactions can be validated for any module
2	Transactions can contain a dynamic amount of other transaction validations in them
3	Transactions can be signed with their sender's private key
4	Transactions can be validated with their sender's public key
5	Equivalent transactions can be squashed together without losing information

Circular Blockchain

#	Test Description
1	Transactions can be added to the blockchain
2	Blocks can remove transactions that are deemed invalid by future transactions
3	Block can overwrite old blocks like a circular buffer
4	The blockchain can be validated by running through each transaction in the circular buffer and validating that they are based on valid info, can be run, and any dependent transactions can have their hashes retrieved from the garburator.

Entanglement

#	Test Description
1	Transactions can be added to the entanglement
2	Transactions validating previous transactions only can validate transactions who's validated transactions were deemed valid
3	Once reaching a certain capacity, it can take all transactions that have enough confirmations to be considered valid, and hand them to the gate

Gate Block

#	Test Description
1	Can receive a bunch of unordered transactions from the lower entanglement
2	Can squash transactions without losing information

3	Can feed squashed transactions over to the Testament block

Testament Block

#	Test Description
1	Can receive squashed groupings of transactions
2	Can squash the squashed groupings of transactions to simplify the data further

Behaviour Checks

#	Test Description	Expected Result
1	Transactions that come in with validation work that does not match the majority of other transactions agreed upon validation work	Rejected
2	Transactions come in with validation work that does match the majority of other transactions agreed upon validation work	Accepted
3	Transactions come in, but then the majority of other transactions validating it claim it is not valid	Rejected
4	Transactions come in, but it validated 'work' for transactions that have already been either been fed to the garburator or squashed	Do not confirm the transaction, however if other users confirm it, trust it
5	Transactions come in which is signed by an address in a module's trusted node list	Accepted, instantly trust the confirmation work shown
6	Transactions come in which validate work for transactions that do not belong to the same module	Accepted
7	A user connects to the network for the first time, and requests to be brought up to date.	User is given the garburator block, circular blockchain data, entanglement and testament block logic, can recreate the world accurately

Details about estimated milestones

Timeline	Milestone	Time Estimation	Total
Jan 4th to Jan 18th	**Sprint 1** Cryptographic Component • Design • Write Tests • Write Implementation (Private key generation, hashing between forms of keys, transaction signing) Seed System • Design	**Design**: 8 hours **Tests**: 5 hours **Implementation**: 40 hours	58 Hours

Jan 19th to Feb 1st	**Sprint 2**
	Seed System
	• Write Tests
	• Write Subsystem Implementations
	  ○ Temporary Transaction Chain
	  ○ Permanent Transaction Chain
	Module System
	• Design
		Design:
		4 hours
		Tests:
		8 hours
		Implementation:
		40 hours
			62 Hours

Feb 2nd to Feb 15th	**Sprint 3** Module System • Write Tests • Write Implementation • Write Documentation	**Design**: 4 hours **Tests**: 4 hours **Implementation**:	48 Hours

	Seed Module • Design	40 hours	
Feb 16th to Feb 29th	**Sprint 4** Seed Module • Write Tests • Write Implementation Relay Node / Networking Component • Design	**Design**: 6 hours **Tests**: 6 hours **Implementation**: 40 hours	52 Hours
Mar 1st to Mar 14th	**Sprint 5** Relay Node / Networking Component • Write Tests • Relay Node Implementation (Nodes that users can connect to as an entrance point to the network) • Network Implementation (JS Clients connection code for communicating with relay nodes & each other) Integration • Integration Design	**Design**: 10 hours **Tests**: 10 hours **Implementation**: 40 hours	60 Hours
Mar 14th to Mar 28th	**Sprint 6** Integration • Write Tests for integration, confirm all previous tests still pass • Integrate Stress Test • Stress Test Network	**Tests**: 30 hours **Integration**: 20 hours	50 Hours
Mar 29th to Apr 11th	**Contingency Sprint** • Fix, Repair and Retest any parts that failed after integration or are behind schedule • Polish any components which are not finished • If time permits after finalizing the project, begin Release stage	**Contingency**: 50 hours	50 Hours
Apr 12th to Apr 25th	**Release** • Final Testing • Final Documentation • Final Report	**Testing**: 10 hours **Documentation**: 10 hours **Report**: 30 hours	50 Hours
Estimated Total Time			430 Hours

Detail all deliverables

Seed: Protocol / Whitepaper

A flexible decentralized server protocol that can be implemented in any language, on any platform. This documentation will showcase the protocol in detail, explaining exactly what must be done for another client to implement the same protocol.

Seed: JavaScript Library Implementation

A JavaScript implementation of the Seed protocol. This library can be included by any JavaScript client which supports Node.js, allowing them to connect to the network, send transactions and utilize modules.

Seed: The Cryptocurrency Module

The first module to be released onto the Seed ecosystem is the Seed cryptocurrency. Code for this module will be delivered, as it is the default module which gives this ecosystem it's first use case.

Final Report

The final report on the progress, development and results of the project.

Development in Expertise

While I have developed basic smart contracts for the NEO cryptocurrencies' ecosystem, and attempted at developing a Python cryptocurrency implementation in the past, this project will further increase my understanding of blockchain technology due to it's sheer depth and complexity. Blockchain technologies have been a passion of mine for years, but I've never had an opportunity to implement my ideas to the extent of this project. The project scope is large; however, I firmly believe I have the skillset, capabilities and learning capacity required to succeed in creating this system.