CISC 322 A1 Report: Conceptual Architecture of Bitcoin Core

Group 27 - Based

February 2023

Amy Cui (19ayc1@queensu.ca)
David Courtis (20dhc@queensu.ca)
Jagrit Rai (19jr28@queensu.ca)
John Alajaji (18ja19@queensu.ca)
Logan Cantin (logan.cantin@queensu.ca)
Matthew Vandergrift (19mwv1@queensu.ca)

Abstract

In this paper we will be exploring the conceptual architecture behind Bitcoin Core by mainly considering the system's various components and interactions, how the system evolves over time, system concurrencies, the flow of control and data throughout the system, as well as considering some important use cases.

1 Introduction

The idea behind bitcoin was first introduced in 2008 in a paper by Satoshi Nakamoto titled "Bitcoin: A Peer-to-Peer Electronic Cash System" [2]. The main idea presented was that Bitcoin could offer a way in which online transactions could safely be transmitted directly from one user to another, without requiring a financial institution to act as a trusted third party. The way that Bitcoin circumvents this issue is by basing the validity of a transaction on cryptographic proof, rather than in trust. This creates a hard, unfoilable currency system.

More specifically, in order for a transaction to be deemed as valid and hence accepted by the community, a certain amount of work must be done in finding a special number (called a *nonce*) which when hashed using a cryptographic function returns a number that is sufficiently small (determined by the target difficulty). Since cryptographic functions are difficult to invert, this process can require a significant amount of CPU time. We will see that this ensures that the network remains secure, as long as fraudulent actors do not control a majority of the CPU power in the network.

In this project we will specifically be considering *Bitcoin Core* which is the reference implementation of the Bitcoin protocol and which serves as the backbone of the Bitcoin network. We will be exploring its conceptual architecture by taking a look at its architectural style, the main components of the system and how they interact, how the system evolves over time, system concurrencies, the flow of control and data throughout the system, as well as considering some important use cases. For use cases we will specifically be focusing on the use cases pertaining to making a transaction, and mining for bitcoin, as we believe these to be the two most important use cases of the system. We begin by going over the process we followed in determining the system's conceptual architecture.

2 Derivation Process

To derive the general architecture of Bitcoin Core along with its various components, we began by doing the required research in order to become more familiar with the system. This included understanding the paper "Bitcoin: A Peer-to-Peer Electronic Cash System" [2], as well as the Bitcoin Core Developer's Guide [1]. In a sense the Developer's Guide was most useful to us, as it pertained specifically to Bitcoin Core, which is the focus of this project.

Once this was done, we began brainstorming ideas about the possible components of the system. It was clear from the start that the architectural style that Bitcoin Core follows was peer-to-peer (this is really the whole point of bitcoin to begin with), however identifying the various components which make up the architecture was not so clear. First we tried to consider all the possible main use cases of the system. In a later section of the report, we will be covering the use cases of making a standard transaction, and of mining Bitcoin, however many other use cases in fact exist. They also include making a bitcoin contract, searching for peers on the network, making various non-standard transactions, and many more. Then for each of these use cases, we considered the various components which are needed in order to ensure that the use case can be carried out.

3 Conceptual Architecture Overview

This section will be dedicated to better understanding the various components present in conceptual architecture

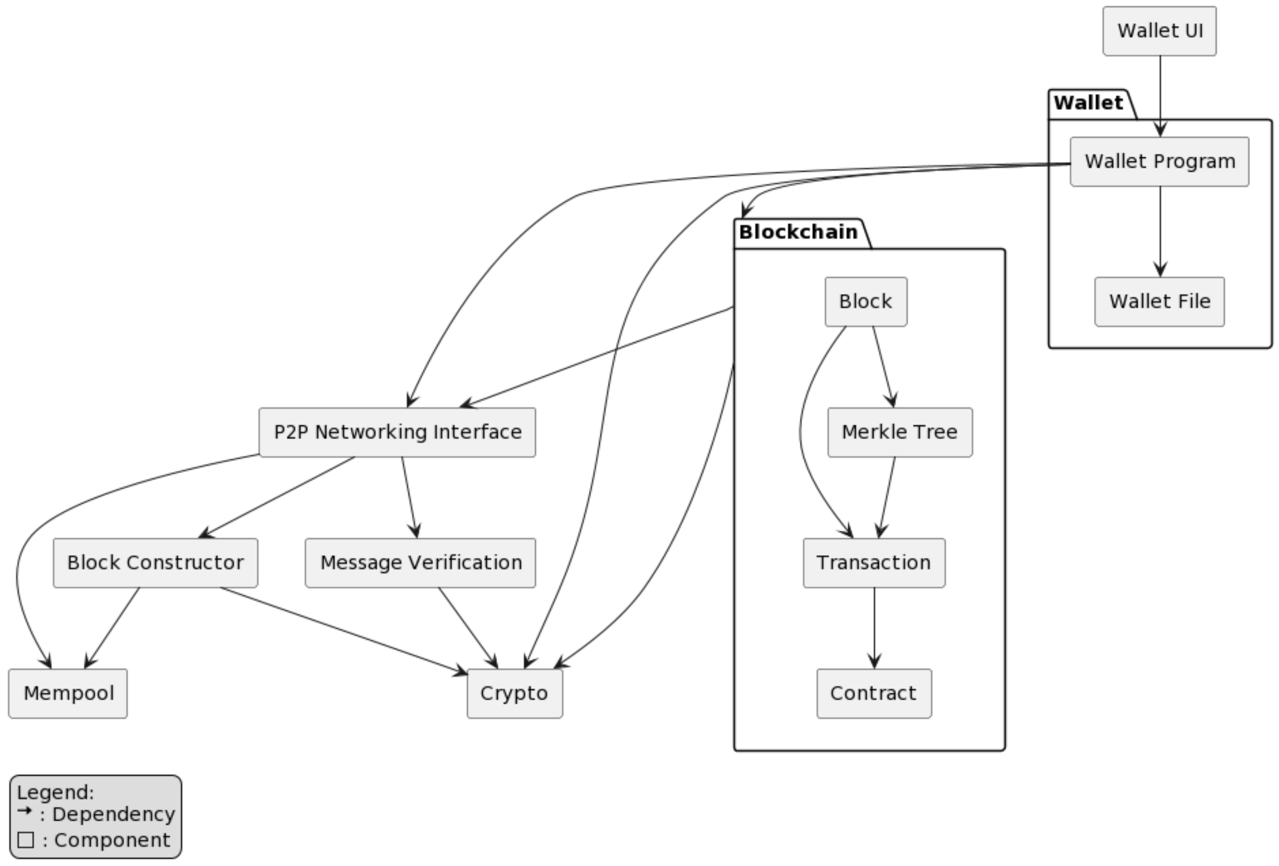


Figure 1: General Overview of a given node in the Bitcoin Core conceptual architecture

of Bitcoin Core. Firstly, before delving deeper into the components of the system, it is important to understand that Bitcoin Core follows a peer-to-peer architecture, in which each node represents an independent part of the overall network. These nodes are connected to each other to form the Bitcoin network; notably, there is not a central component through which all network traffic flows. The fact that each node runs independently, providing data and services for each other over the network without any centralized control fits exactly with the definition of peer-to-peer style given in lecture ([5] page 155). The consequences of this choice of architectural style will be expanded upon in further sections, but for now the key idea to retain is that the system is fully comprised of nodes (representing the various users), and within each node is housed the components which enable Bitcoin Core to function.

Figure 1 presents a general overview of the various components of a node in the Bitcoin Core system, and demonstrates the dependencies between components. As is visible from this figure, the components of a node are broken up into a few core parts. Firstly there is the Blockchain, which itself is comprised of Blocks in which there are Transactions and a Merkle Tree (as well as possibly contracts which transactions can depend on). There is then also the Wallet subsystem, which is comprised of the Wallet Program (acts as a transaction manager), as well as the Wallet File component (responsible for actual storage of transaction keys). Finally there are the other components of the node. These range in functionality, however their key attribute is that they typically tie back into one of the previously mentioned subsystems, and ensuring their proper functionality. For example, the P2P Network Interface is needed to fetch the blocks from the network to be added to the Blockchain.

4 Components

4.1 P2P Network Interface

The Bitcoin network has a peer-to-peer (P2P) architecture at the highest level. Importantly, that means that there isn't a central server that orchestrates the entire network: all communication happens in an ad hoc manner between peers in the network. The P2P network interface (also commonly referred to as bitcoind in the literature) is the main point of contact between a node and the rest of the network. Its role is to listen to the networking hardware for relevant messages, process them, and rebroadcast when required by the Bitcoin protocol.

A major role of the network interface is to discover and maintain a list of a few peers so that it can stay connected to the network. If it is determined that a peer is malicious, usually because they are found to be sending messages that don't pass the message verification, that node will be blacklisted for a period of time to punish delinquent behaviour on the network.

4.2 Message Verification

There are a variety of verification checks that are done to the messages that are received, both for security and network health reasons.

- Block / transaction verification: A significant amount of network traffic has to do with the broadcasting of proposed transactions and newly minted blocks. However, since this is a P2P network, it is possible that neighbours are malicious and are sending false information. In order to prevent these false transactions and blocks from hogging network bandwidth and CPU time, a neighbour will evaluate the validity of a block or transaction before adding it to its Blockchain or rebroadcasting it.
- Broadcast storm prevention: Messages are given a unique id based on their contents and stored in a local database. This way, if a message gets rebroadcast and then received again, this system can catch it before it loops and multiplies, consuming all the bandwidth and stopping operation of the network.
- Packet verification: This system checks the packet to make sure it is well formed. This is a security feature to prevent adversaries from crashing the nodes or sending malicious packets.

4.3 Cryptographic Component

The original paper described Bitcoin as an "electronic payment system based on cryptographic proof instead of trust"[2], which highlights how important cryptographic functions are to the operation of Bitcoin. There are essentially two classes of cryptographic functions that are used:

- **Cryptographic Hashes:** A hash function is a function that takes an arbitrarily large and outputs a number of a fixed size. Cryptographic hashes have some special properties that make them useful for cryptography; notably, they are hard to reverse (i.e. given an output, it is hard to find an input whose hash equals the given output) and it is hard to find collisions (i.e. given an input message, it is infeasible to find another message that has the same hash). Cryptographic hashes are used in the Bitcoin protocol in two key places. First, they are used for proof of work. The fact that hashes are hard to reverse means that the only way to mint blocks is by doing a significant amount of computation. Secondly, hashes are used to uniquely identify different blocks on the Blockchain.
- **Digital Signatures:** Asymmetric encryption is often used for making digital signatures. It works by having a public and private key; the private key is used to "sign" a piece of data, and anyone can use the public key to verify the signature. It is computationally infeasible to forge a digital signature, so if you verify a piece of data using the public key, you can be pretty certain that it was signed by the private key holder. In the context of Bitcoin, digital signatures are used in transactions to verify that the owner of currency approves of the transfer of funds.

This component is responsible for generating all key pairs (for asymmetric encryption) and doing all cryptographic calculations, such as hashing or verifying digital signatures.

4.4 Blockchain

At the core of Bitcoin is the notion of the Blockchain, which is a decentralized ledger system. As the name implies, it is comprised of blocks which are connected to one another (described in 4.4.1). Through connections between blocks, you can trace a chain of blocks back to the root and verify the chain of information. This makes it incredibly difficult to alter blocks that occurred earlier in the Blockchain (such as by backdating transaction data), as you would need to update every subsequent block in the chain, which becomes increasingly computationally expensive as you go further back in the chain (to the point of being infeasible after about 100 blocks or 16 hours).

4.4.1 Blocks

A block is a key component in Bitcoin Core's implementation of the bitcoin protocol. The block is a collection of transactions (called transaction data) preceded by a block header, which tracks the ownership and transfer of Bitcoin over time. A single block can store zero or more transactions. Each transaction itself is digitally signed to prove its validity. The block header contains the hash of the previous block header, as well as a Merkle root (4.4.4), which is a single value that hashes the unique hash of each transaction on the block, and a nonce, a value that is used for proof of work.

4.4.2 Transactions

The transaction part of Bitcoin Core is what allows users to exchange Bitcoin. Transactions could be more aptly described as a subsystem than a part, since currency exchange is represented through a system of interlinked transactions. The link is established through connections between the data fields of a transaction. The data fields are inputs, outputs, a version number and a locktime.

- Inputs/Output: Each transaction is an object that consists of several inputs and outputs. The input is where the bitcoin is coming from and the ouput is to whom it goes. An input of a transaction will use the output of a previous transaction. Newly created outputs are saved as an unspent transaction output (UTXO), until a later input uses it, rendering further uses of this output invalid. When a party wishes to check the balance of their wallet, they are really checking the sum of their owned UTXOs.
- Version: The version of a transaction specifies the set of rules to validate it. This is a feature to allow for future implementation of a new validation method without the need to apply the method to all previous transactions (thereby invalidating them).
- Locktime: The locktime of a transaction is the soonest possible time point at which a transaction can be linked in the chain. This means that not all transactions are immediately put onto the chain. This is useful for any delayed transactions, for instance if users want to have a chance to reconsider an exchange.

4.4.3 Contracts

A contract is part of a Bitcoin transaction used to structure agreements through the block chain. The term contract is being used in this report to refer to the set of rules which the two parties involved must agree upon before a transaction can be sent. These rules typically have to do with altering the input and output data of the transaction block. Users can specify that both users need to sign the transaction, that a third party needs to sign, or that a certain condition must be met before funds exchanged. The realisation of these rules are controlled by the following two sub-components:

- *Pub-key Script*: The instructions which state the conditions required before the exchange of coins can occur.
- *Signature Script*: The data or parameters used to determine if the conditions have been meet.

4.4.4 Merkle Tree

A Merkle tree is a component of the Blockchain which is used for optimization purposes, and it facilitates travel around the Blockchain for the purpose of verifying old transactions. Instead of requiring miners to linearly search the Blockchain for an old transaction, they can jump directly to the corresponding block.

As shown in **Figure 2**, this root hash is a unique representation of all the transactions in the block and serves as proof that the transactions were included in the block. If any transaction is mutated, the change will propagate up the branch and toward to root, thereby invalidating the mutating action.

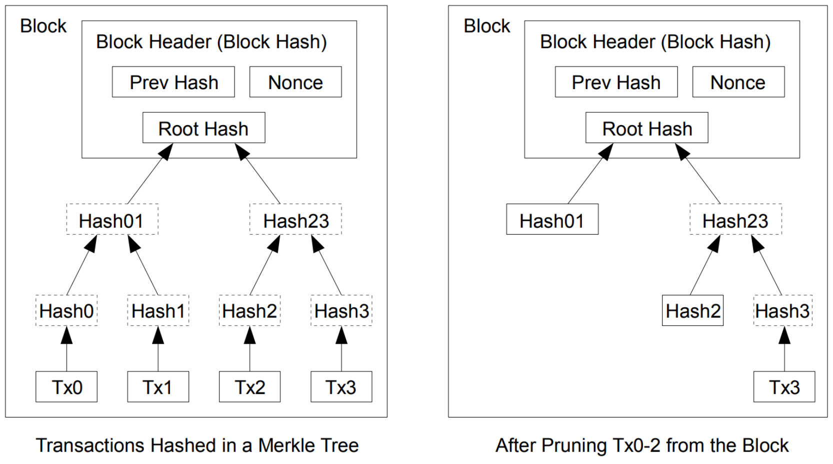


Figure 2: The Merkle tree of a block, with the header containing only the root, and verification possible with only the relevant nodes of a branch. [2]

4.5 Block Constructor

The Block Constructor is responsible for the minting of new blocks during the mining process. To do so, using the *getblocktemplate* remote procedure call it is provided a pool of transactions obtained from the Mempool component (4.6), which it then uses to construct a block along with a block header. Note that *getblocktemplate* RPC will also provide the Block Constructor with the information necessary to construct a coinbase transaction (which will be included at the start of the block), other information needed to construct the block header (version number, previous block), as well as the target value D for the block's proof of work.

Next, the Block Constructor then utilizes the cryptography component to generate a proof of work for the block. This is done by attempting to hash different nonce values for the block until the overall hash of the block is less than a target number D. In the case of solo mining, D is in fact the network difficulty, and is computed such that it takes about 10 minutes for a new block to be minted. This nonce is then recorded in the block header, as proof that CPU work went into building the block.

4.6 Mempool

This component queues incoming valid transaction, and by request provides them to the Block Constructor to construct a new block during the mining process.

4.7 Wallet

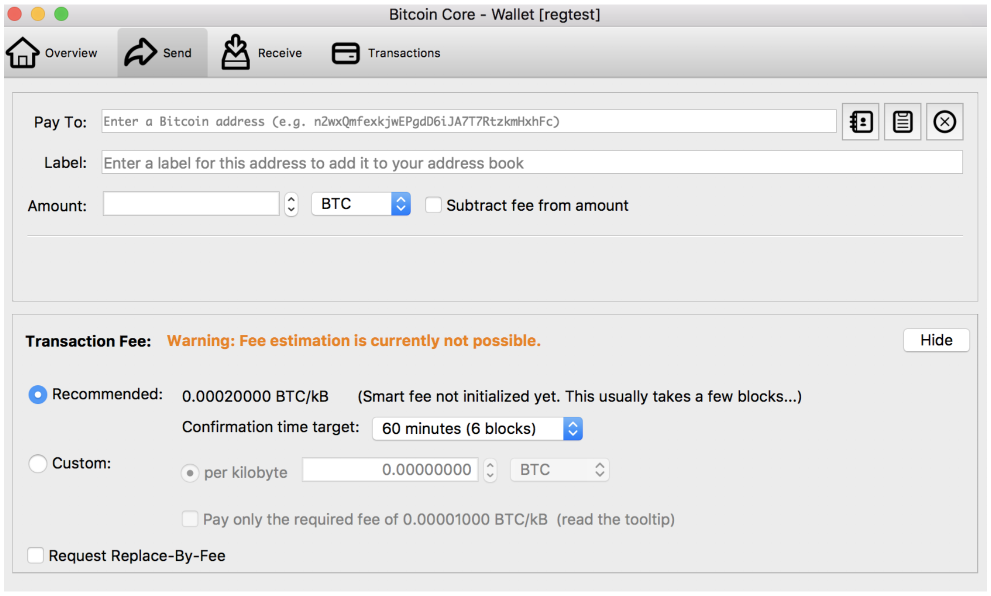
A Bitcoin wallet is composed of the wallet program, wallet UI, and wallet file components. Essentially, the wallet program is the user-facing component of a Bitcoin wallet that provides the interface and functionality for managing funds. The wallet file on the other hand is the backend component that stores the critical information needed to access and manage the funds. The wallet UI allows the user to interact with the system.

4.7.1 Wallet User Interface

Bitcoin Core's wallet user interface (Wallet UI) component allows users to send, receive, and monitor their transactions. It also allows the user to get an overview of their wallet, giving the user easy access to their balance, transaction history, and other relevant information. **Figure 3** shows the Bitcoin Core's Wallet UI for the case where a user is attempting to send a transaction.

4.7.2 Wallet Program

A wallet program is a software application that enables users to securely store, manage, and transact with their Bitcoin. It accomplishes this by creating public keys to receive Bitcoins and use the corresponding private keys to spend those Bitcoins. The public key serves as the receiving address for Bitcoins and can be freely shared with



Pay To: Enter a Bitcoin address (e.g. n2wxQmfexkjwEPgdD6iJA7T7RtzkmHxhFc)

Label: Enter a label for this address to add it to your address book

Amount: BTC ☐ Subtract fee from amount

Transaction Fee: **Warning: Fee estimation is currently not possible.**

● Recommended: 0.00020000 BTC/kB (Smart fee not initialized yet. This usually takes a few blocks...)

Confirmation time target: 60 minutes (6 blocks)

○ Custom: ● per kilobyte 0.00000000 BTC

☐ Pay only the required fee of 0.00001000 BTC/kB (read the tooltip)

☐ Request Replace-By-Fee

Figure 3: Bitcoin Core's Wallet UI

others to receive payments. On the other hand, the private key is kept confidential and is used to sign transactions while also providing proof of ownership of the funds associated with the corresponding public key. Overall, the design of a Bitcoin wallet plays a crucial role in ensuring the security and usability of the Bitcoin ecosystem, and is a critical component of the overall software architecture of the Bitcoin network. There are four main types and two subtypes of wallet programs that users can choose from, each with its own unique features. The Bitcoin Wallet program has many functionalities. It is responsible for generating and managing the public and private keys of the wallet. It also allows the user to import, export, backup keys and create new addresses to receive Bitcoins. Additionally, it enables users to initiate and monitor transactions, view transaction history, and manage the wallet's balance (through the Wallet UI). It also communicates with the Bitcoin network to retrieve information about unspent transactions and broadcast transactions. The Wallet UI (section 4.6.1) provides an interface for the user to interact with the wallet, and the P2P Network Interface (section 4.1) component facilitates communication between the wallet and the Bitcoin network.

4.7.3 Wallet File

Another key component of the system is the wallet file component. A wallet file is a digital file that stores the private keys of a Bitcoin wallet. Wallet files are typically stored digitally in a file such as the user's local computer or device, or can even be physically stored on pieces of paper. There are three key formats that the wallet file consists of. One such format is the Wallet File Format (WIF) which encodes private key information in a secure manner, decreasing the chance of copying error. Wallet files are encrypted (since version 0.4, as discussed in section 6) to protect the private key information from unauthorized access. Key management is another essential feature, which allows users to import and export private keys, as well as create new Bitcoin addresses. Finally, the wallet file is stored either on the user's local device or can be physically stored, providing a safe and accessible storage location for private keys.

5 Noteworthy Use Cases

5.1 User Transactions

In this common use case, a user (the sender) wants to send another user (the receiver), X amount of currency. Figure (4) is a sequence diagram for this use case. Before the sender can transfer X quantity of currency, they need to know where to send it. Transactions are sent to a public key hash, colloquially referred to as a bitcoin address. The first action on the sequence diagram is therefore the passing of a bitcoin address from the receiver to the sender, this happens outside of the bitcoin core node since it could be exchanged by any method. Once the

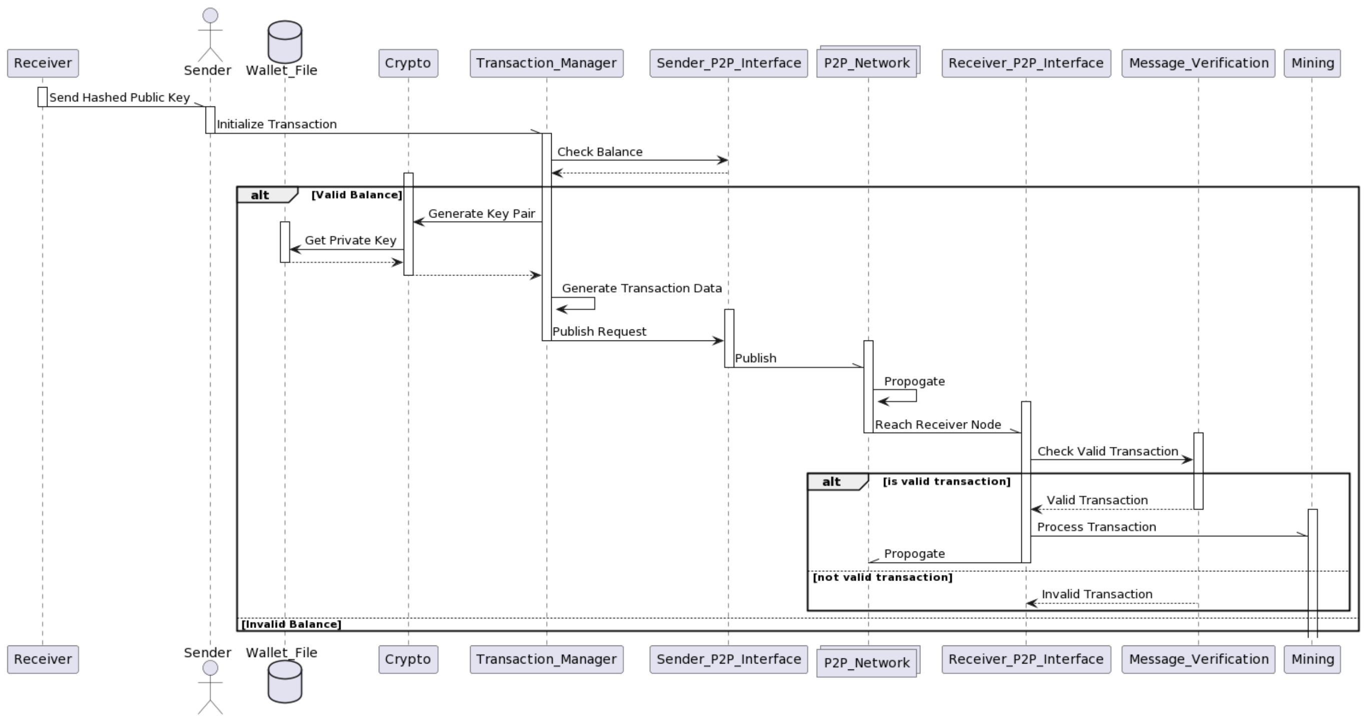


Figure 4: Sequence diagram for a user transaction use case

sender knows where to send their currency, they will specify the details of how much they want to transfer to this address in the UI, and this information will be sent to the transaction manager. An evident condition for transfer is sufficient balance. A user's balance, which is stored as the sum of UTXOs, can be found by looking at the public transaction data on the block-chain. As such the transaction will pull this information by calling the peer-to-peer network interface and if the user has sufficient balance the transaction process can continue.

In order for a transaction to be valid, the signature script of its inputs must hold parameters that satisfy the Pubkey script of the outputs. This pubkey script specifies the conditions and processes required for validation.

For the receiver to spend UTXOs, the receiver must create a new transaction which references the transactions the spender created by their hash, the specific outputs by their index, and a signature script that satisfies the pubkey. The signature script will contain the complete public key, and a signature, comprised of a cryptographic formula that combines a private key with various parts of the transactions, to verify that the receiver owns the complete public key.

To allow this interaction, the sender will generate the transaction output implemented as a pubkey script (4.4.3). To facilitate the key validation process, the transaction manager will call a cryptography component that generates a private-public key pair. It is worth noting that some required information is determined in part on the sender's private key necessitating a wallet file call to obtain the sender's private key. With the private key, the rest of the transaction data can be generated. This data includes the locktime, the inputs, and outputs, and the version (more information on these fields found in the components section).

Once the transaction data has been generated, it can be sent out onto the network. To do this, the sender's peer-to-peer interface will broadcast it to a few nodes (8 is the current specification), then these notes will propagate to their connected nodes. This process is represented in the sequence diagram by the collection component P2P_network which represents other nodes re-propagating the new transaction. This transaction will reach receiving peer-to-peer interfaces, which will check the message to ensure validity. If the message is valid then the receiver will indicate this and the transaction can be sent to the mempool to be mined. The mining of a transaction is described in detail as the next use case.

5.2 Mining

In this section we will describe the use case of mining bitcoin. We chose this use case, since aside from making transactions, it is one of the most common use cases, and is essential in order for transactions to be added to the

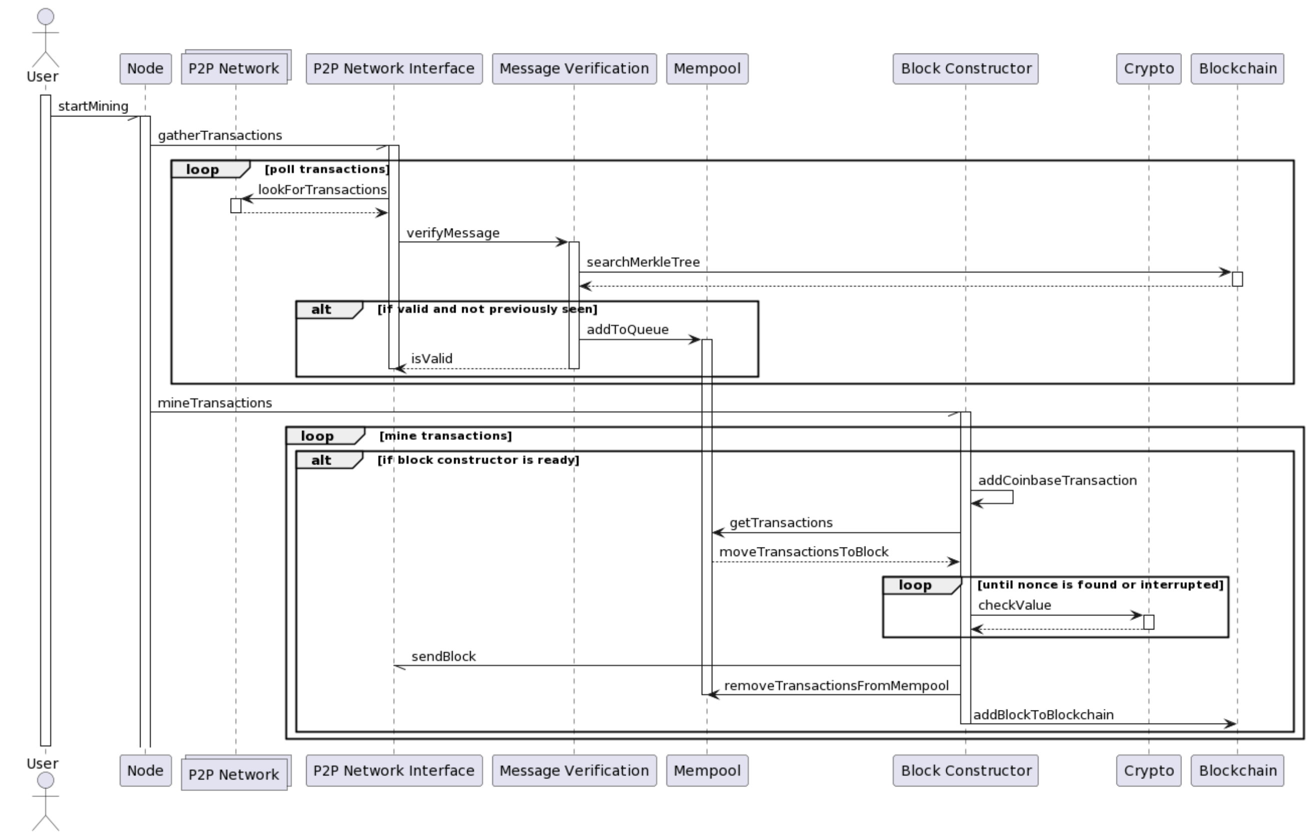


Figure 5: Sequence diagram for the solo mining use case

Blockchain. For the sequence diagram pertaining to this use case, refer to **Figure 5**.

Mining bitcoin is the action in which individuals group together broadcasted transactions into blocks, to then be added onto the Blockchain. However, in order for the block to be valid, they must first generate a proof of work which is done by generating a special number (called the *nonce*), which when input to a cryptographic function, the nonce gets hashed to a number with a predetermined number of zeros (which determines the difficulty of the proof of work) [2]. Since cryptographic functions are difficult to reverse engineer, this step requires the miner to spend processing time verifying random input values until a valid output is obtained. This ensures that bad actors are not able to reliably take control of the Blockchain as long as they do not control more than half of the processing power in the network.

It is important to note that mining bitcoin can take one of two forms. Firstly there is *Solo mining* in which an individuals acts independently to mine bitcoin, and then there is also *Pool mining* in which a group of individuals pool together their resources to mine bitcoin as a unit. These two methods of mining only vary slightly, and differ only by the fact that in pool mining, there is a central node which acts as a Pool Manager and which coordinates the mining strategy. However, in this section we will only be covering the case of solo mining.

For a user to begin mining a set of transactions, the user first alerts the P2P Network Interface to begin polling the P2P network (a collection of other nodes in the network)) for new transactions. Received transactions are then sent to the Message Verification component to verify their validity (which also checks the Blockchain to accomplish this). Invalid or previously seen transactions are discarded, whereas the valid transactions are then sent to the Mempool, a transaction pool which stores them until the current block is mined by an arbitrary miner on the network (the valid transactions are also propogated, but this is detailed in the transactions section). Once the Block Constructor is available, it then constructs a block header, and a block using the queued transactions in the Mempool. It is important to note that the first transaction included in the block must be a *coinbase transaction* in which new bitcoin is minted and given to the miner(s) (however it cannot be spent for at least 100 blocks, to prevent a miner from spending a block reward from a block which ends up not being added to the Blockchain [1]). The miner's transaction fee which is obtained from the difference in inputs and outputs of transactions in the block should also be included in the block.

The Block Constructor then constructs a proof of work. This is done by constantly communicating with the

Crypto Component to iterate over possible nonce values, and checking if that value hashes (via a cryptographic function) to a valid target. Once the nonce for the current block is found by a miner, the mining operation for other miners are halted. The nonce is then attached to the block header, which terminates the block construction process, and the fully constructed block is sent out over the network (via the P2P Network Interface), as well as added to the local version of the Blockchain. Note that this propagation will also be received by wallets for their UTXO ownership update. The transactions used to construct the block are then removed from the Mempool, and the next set of queued transactions are used to repeat the mining process.

6 System Evolution

In this section we will cover how the Bitcoin Core system has evolved to get to where it is today. The oldest version of the system which has been documented was simply called Bitcoin version 0.3.21. This is not necessarily the original version of the system, however is the earliest documentation made on the Bitcoin Core version history page [3]. It appears that the key feature introduced in this version was to allow for full-precision bitcoin amounts.

In Bitcoin v0.4, the main change was encryption of wallet private keys. This meant that users could now choose to add a passphrase to access their wallet, but losing that passphrase meant they would lose all their bitcoins. Since previous versions of the client were unable to read encrypted wallets, they would crash upon startup if a wallet was encrypted. Due to the peer-to-peer architecture of Bitcoin, it makes sense that backwards compatibility was not a requirement of the software, instead all users in the network are encouraged to run the same version and follow the same rules to obtain full functionality. This allows for the widespread implementation of such important changes.

The 2011 Bitcoin-Qt v0.5 update came with few functional changes, however made major changes to the graphical user interface, totally focusing on user experience asides from the typical bug fixes. Some of these changes included drag and drop functionality for certain files, and showing loading progress bars for network synchronization and wallets. This update shows an interest from the developers to reach a broader, perhaps less technical audience. Bitcoin-Qt version 0.5.3 introduced an important protocol update: the network rule which dictates that "a block is not valid if it contains a transaction whose hash already exists in the block chain, unless all that transaction's outputs were already spent before said block" beginning on March 15, 2012. Bitcoin-Qt version 0.6 went on to introduce displaying and saving of QR codes for sending/receiving addresses. This change allows for better everyday use of the software, further exhibiting the pattern of the development towards widespread use.

In March of 2014, The software was rebranded to 'Bitcoin Core' as to reduce confusion between the Bitcoin software client, which serves to connect a computer to the Bitcoin network, and the Bitcoin network itself. This update publicly disavowed storing non-currency related data in the Blockchain, as the developers noted that it was "less costly and far more efficient to do so elsewhere". Another notable change was to bitcoind (the P2P Network Interface). Bitcoind formerly acted as a server and remote procedure call client– the latter was separated from bitcoind in this update. This change in functionality is important to note, as bitcoind plays an important role in the architecture of the Bitcoin Core system as an interface between the software and the underlying network.

Version 11.0 of Bitcoin Core then introduced block pruning, which allowed for some redundant information to be cut from the blocks (to save space). However it is important to note that changes such as these can actually have consequences, and end up affecting other components of the system. In this case, this new feature made running a wallet incompatible "due to the fact that block data is used for rescanning the wallet and importing keys or addresses."[3] This demonstrates the importance of considering the dependencies between components, in order to understand how an update may affect the other components of the system.

Finally, the most recent version of the software known as Bitcoin Core 22.0 has also made several changes, however most noteworthy was some changes made to the P2P Network. That is, support for running Bitcoin Core as an Invisible Internet Project (I2P) service was added, increasing user privacy by adding an extra level of anonymity to the peer-to-peer communications. This is the current version of Bitcoin Core, which is available for download today.

7 Control and Data Flow

Bitcoin Core is composed of multiple layers and components, each of which performs a specific task. This section focuses on the control and data flow of Bitcoin Core components in more detail, focusing on how transactions and blocks are processed and propagated through the network.

The control and data flow of Bitcoin Core describes the order in which instructions are executed, and movement of information between various components. These descriptions are influenced by the program's design, user input, and the network's state.

7.1 Transaction flow

The utility of Bitcoin Core stems from its use as a currency. Beginning from when a user initiates a transaction and ending when the transaction is minted to the ledger. There are several stages in the minting of a transaction.

When a user initiates a transaction, the transaction is first published to the Bitcoin network and verified as legal by the nodes on the network. Each node ensures the inputs and outputs of the transaction balance and that the transaction is properly signed. If it's valid, the transaction is added to the node's mempool, which is a list of unmined transactions that have been verified by the node, to be processed. In the case that the network is congested, higher-priority transactions (i.e. those with higher transaction fees) are processed first.

A set of transactions are selected to be processed in the next block, and the node performs a set of checks to ensure that the block is valid, including if the block is properly linked to the previous block in the chain, and that the block meets the network's difficulty requirements. When a miner creates a new block, they include the coinbase transaction as the first transaction in the block. Once a block is validated, it is broadcast to the network's other nodes and added to the Blockchain.

7.2 Blockchain Flow

The driver behind the validation and minting of transactions are the miners and nodes of the network. These agents ensure the integrity of the Blockchain.

When a new block is broadcast to the network, nodes receive it through the P2P network interface and begin a set of checks to ensure that the block is properly constructed, the most recent, and contains a valid proof-of-work solution. Nodes also verify that the transactions in the block are valid and that they have not been double-spent or otherwise tampered with by bad actors. Once a block has been validated by a node, it is merged to the node's locally stored Blockchain and the transactions in the block are removed from the node's Mempool. The node re-broadcasts the new block to other nodes in the network.

In the event that someone wants to verify the inclusion of a specific transaction in a block, they only need to be provided with the transaction hash, the corresponding intermediate hashes, and the root hash. With this information, they can perform a small number of hashes to verify the inclusion of the transaction in the block and ensure that the data has not been tampered with. This information can be obtained by a full peer, a peer that holds the entire Blockchain, including the full transaction data.

While the nodes are listening and acting upon changes to the chain, miners concurrently gather sets of valid transactions from their Mempool, and then attempt to mint a new block by doing the proof of work. Once a block has been mined and validated, it is added to the miner node's local Blockchain and the transactions in the block are similarly removed from the node's Mempool. This block is now broadcast to the network for propagation.

8 Concurrency

The P2P architecture inherently adopts components that interact asynchronously. We opted to split the discussion on concurrency into two sections: one focusing on intra-node concurrency (concurrency within a node), and the second on inter-node concurrency (concurrency between nodes in the network).

8.1 Inter-node concurrency

Bitcoin Core allows for a large degree of possible concurrency. Nodes are never locked out from performing actions due to having to wait for other nodes to finish. A quintessential feature of a peer-to-peer network is transient or temporal connections. There is never a guarantee of when or for how long a connection to the network might exist. A consequence of this is that the degree of concurrency is inconsistent. The number of nodes performing actions on the network at once will vary. Because nodes do not block other nodes, any increase in nodes will lead to an increase in concurrent actions.

One specific example of a task that is handled concurrently is networking. When a message is sent to the network, it is sent to the peers with a "best-effort" strategy. That is, it is sent off to the peers, but the node does not verify that it reaches every node in the network. In this regard, every node does its own networking independently from the other nodes, following the P2P networking rules. This contrasts with the server/client architecture, where the server is responsible for all networking between peers.

Another example of inter-node concurrency is mining. Especially in solo mining, nodes do their proof of work individually from each other, only syncing up once a block is mined. In pool mining, there is a higher degree of synchronization to ensure that the miners in the pool don't check overlapping nonces, but overall there is still a high degree of concurrency.

8.2 Intra-node concurrency

Not every component in the node operates concurrently, however there is still a few key examples of intra node concurrency. The User Interface has a lot of intra-node concurrency. When a user is interacting with bitcoin core they might wish to check their balance send money to someone, view their transactions history etc. As discussed in the use case section, sending a transaction is a large procedure which will take time, however the UI will not stall while this occurs. The UI will concurrently handle user inputs and queries while passing them to whatever components can handle the specific inputs. This is fairly typical for a UI and would be expected by most users.

The Block Constructor will operate concurrently to other components. This is because the block constructor is a principle actor in mining which is designed in the bitcoin protocol to be a time consuming process. It would be impractical for the full node to shut down when the node is doing any mining. If someone is mining they are still able to send/receive transactions, view their balance, and sync with longest chain. Additionally multiple cores of the computer running the core can be working on the same nonce at once. This is a way in which concurrency can happen just within the block constructor.

The concurrent operation of the peer to peer interface is crucial to the operation of the full node. As outlined in section (6), the Blockchain is constantly growing and shifting which could impact the actions of a full node. For example if a node is attempting to find a hash for the nonce, and then the difficulty is updated it will need to adapt its actions. This implies that the full node must be frequently listening to it's peers on the network for important messages. Furthermore, it must clearly be re-propagating these messages. It would not make sense for the other components to stop frequently to allow for listening and reproprogating. As such we see that components are frequently operating while the peer to peer network interface listens. There is however a caveat which is that if information is received by the network interface with affects the operation of a component it can interrupt that action. This is what happens in the case of increasing mining difficulty.

9 Division of responsibilities among developers

The components already discussed in section 3 present a good way to discuss the various ways in which development responsibilities and skills can be split up.

The first possible division comes with respect to the cryptography component of the system. This part is principally concerned with hashing data efficiently and consistently. Bitcoin core implements complex hashing functions based on high level mathematics, for instance elliptic curve based hashing. The use of complex number theoretic algorithms would suggest that mathematicians or programmers with a background in math would be best suited to work on this component.

Another clear division based on skill comes from the network interface component. The mechanics of connecting a node to minimum 8 other nodes efficiently over a network clearly requires knowledge of computer networks. Given that the peer to peer network is the center of the bitcoin core architecture it would be advisable to give such a component to experienced networking developers.

An less obvious but equally important division comes in implementing the wallet file and offline wallet. These wallet programs store private keys which have the potential to represent massive amounts of currency for a user. This is represents an important security as they might be a target of malicious attacks. Given the decentralized structure of bitcoin the security risk from local attacks on wallets. This is why developers with experience in system level security might be helpful for this task. Ensuring that the wallet is not vulnerable to other programs on a user's computer or to system failures is critical.

The last skilled based division comes the UI component. There exists a profession of developers who focus on UI design and UI implementation. If it is desired to have a UI which meets industry standards for modern software then a UI team/developer will be needed. They need not have bitcoin specific knowledge but rather the knowledge of how to build a good user interface.

Now we discuss how developers working on different components could communicate/integrate their work together. Given the peer to peer nature of the system, all the different components only have to interact within a node. Since it is assumed for this system that all nodes follow the Bitcoin protocol as described in Satoshi's whitepaper, developers only need to focus on connecting their components locally on the node. This means that developers must have agreed on data sharing mechanism, and all must have knowledge of what is stored on the system, and is available to be viewed at any given moment. For instance if unspent UTXOs are not stored in the wallet it is important for the transaction manager developers to know this. They can then implement the ability to get this information from the Blockchain. Given this fact it would seem important for developers to document what is stored on the node and how this data could be passed locally in order for efficient splitting up of development time.

10 Architectures and Alternatives

Bitcoin core was built with specific architecture designed to achieve decentralization, security/privacy, and resistance to censorship. These goals are advantages of a peer to peer network, especially decentralization and privacy ([5] page 161) As mentioned in the conceptual architecture overview independent peers connected by a network without a server is quintessentially peer to peer ([5] page 155)

The overall architectural style of Bitcoin Core is evidently peer-to-peer; however, the prevalence of the object-oriented style in the implementation of Bitcoin Core's full nodes is less clear. While the implementation of Bitcoin Core indeed utilizes various object-oriented programming principles, such as inheritance and polymorphism, it is primarily written in C++ and uses a mix of procedural and object-oriented styles. Many of the components of a Bitcoin full node can be thought of as objects, such as the transaction object and the block object, and these objects have properties and methods that can be used to perform operations on Blockchain data. Components being objects which are connected via methods (the functions which perform operations), is the given definition for Object Oriented Programming ([5], page 100) hence it is applicable.

It is also important to note that the Blockchain and the wallet file are a form of database in which data is stored for access by various sub-components. These are the common features of a repository architectural style ([5], page 75). The Blockchain is commonly referred to as the ledger and is responsively shared by miners in the node, while the wallet file contains a set of asynchronous local, private, data that is updated by the network, and enables agents to create valid transactions with respect to the network.

We will both explore potential alternative architectures and give insights into the chosen variation.

10.1 Other Architectural Styles

While there are certainly alternative architectures that could be used in the context of transaction management, it likely would represent a significant departure from the principles of decentralization, security, and privacy that are at the core of the Bitcoin network.

10.1.1 Publish Subscribe

This architecture fits naturally with a large quantity of the essential features for bitcoin, although ultimately does not apply. The broadcasting system naturally supports the ad hoc nature of node connections [5] (page 107). Different nodes could chose when to attach to the bus and only broadcast when they have important data. For instance when a node wishes to propose a new transaction it will broadcast it to the broadcasting system, and it would detect which nodes are interested. When a new block is minted this could be published and every node would be interested so it could be sent out to all by the broadcasting system.

This is the core weakness of this architecture: the broadcasting system is a single point of failure. This poses reliability issues since if the broadcasting system ever malfunctioned the block chain might not be accurate. It also posses scalability issues since the broadcasting system would have to increase it's knowledge of what nodes are interested in what as the number of nodes increase. Finally it would add security concerns since any attackers could potentially intercept and modify messages over what would have to be a distributed/network based broadcasting system. The broadcasting system would need to be this way given the wide physical spread of the nodes.

10.1.2 Client Server

This architecture has many aspects that are a perfect fit for bitcoin, and others with are directly contradictory to the goals of bitcoin. The most obvious benefit is the efficient distribution of data ([5] page 152) which for a large networks of nodes would have enormous benefits. No data would be have to ever be duplicated since it could all be stored on a server. Additionally the location of the data would never be in question. In the P2P architecture, a group of nodes could get stuck in a clique and temporarily lose access to new data. A central server would eliminate this risk.

The major downside of a client server architecture would be the loss of decentralization. All currency would be stored in a server which would introduce the exact issues of trust, bitcoin was designed to avoid. Whoever controlled the server would in effect control the bitcoin currency.

10.2 Other Block-chain Variations

10.2.1 Second layer Scalability

As the network is limited to a block, or around 2000 transactions per 10 minutes, various proposals are introducing potential solutions for scaling this limitation. A common example is an alternative Blockchain transaction software called the "Lightning Network" which uses a second layer that operates on top of the main Blockchain. This allows users to send and receive Bitcoin quickly and cheaply without having to wait for confirmations on the main Blockchain. Transactions are instead validated and settled off-chain, with the final state of the transactions being recorded on the Blockchain only when the channel is closed.

An issue when applying this technology to Bitcoin Core is that networks require the participation of second layer nodes that are willing to route payments, thereby causing a clustering of the network by layer. This concentration of nodes could lead to centralization, conflicting with a core principle of Bitcoin Core. Additionally, off-chain transactions are not protected by the full security of the bitcoin network as validation workloads are skewed out of the off-chain.

10.2.2 Proof of Stake (PoS)

Because of the energy-intensive nature of the proof of work variation, it was proposed that only holders of the cryptocurrency are able to validate transactions and create new blocks on the Blockchain. Users "stake" their cryptocurrency holdings as collateral to be chosen as validators, with the probability of being chosen for validation being proportional to the amount of cryptocurrency they have staked, naturally decentivising bad actors.

Natrually, PoS indicates that the probability of being chosen as a validator is relative to the amount of cryptocurrency a user has staked, thereby users with more cryptocurrency have a higher chance of being chosen as a validator, which could lead to centralization of the validation network. An important consideration is when validators simultaneously validate multiple versions of the Blockchain, thereby not effectively staking anything, and leading to potential inconsistencies in the network.

11 Lessons Learned and Limitations of Project

This was a large project to undertake. Despite having 6 people, it was hard to divide the project up and work on it concurrently, because all of the pieces are interconnected and have to be cohesive as part of the bigger picture. We took the approach of dividing up the system first, researching the components and writing the report individually. This approach led to many inconsistencies in the way that we viewed the components and resulted in having to redo sequence diagrams multiple times. Next time, we are going to work together to research and understand the system holistically, and only then work individually once everyone has a shared understanding of all of the pieces.

The main limitations encountered throughout this project are the limited materials from which our analysis was drawn. We mainly relied on two sources: the Bitcoin developer guide and the Bitcoin whitepaper. These sources are trustworthy and contain a dearth of information so it was reasonable to rely on them; however, it did limit the scope of our research. No interviews with developers or experts were conducted or reviewed. This means that when an element of these resources was unclear to us, we relied on either the open source book[4] or an educated guess. If this project were done again, a preliminary step might have been for the group to conduct a brief review of trustworthy literature to address areas of confusion before beginning work on the final product. This would be helpful since our group was not initially familiar with cryptocurrency and hence points of confusion came up frequently. Establishing a common set of references would have added consistency to how we interpreted the complex system of Bitcoin Core.

References

1. “Developper’s Guides.” *bitcoindeveloper*, https://developer.bitcoin.org/devguide/mining.html. Accessed February 12 2023.
2. Nakamoto, Satoshi. “Bitcoin: A Peer-to-Peer Electronic Cash System.” (2008).
3. “Bitcoin Core version history.” *BitcoinCore*, https://bitcoin.org/en/version-history. Accessed February 16 2023.
4. Mastering Bitcoin by Andreas M. Antonopoulos (O’Reilly). Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6.
5. Week 3 Lecture Slides from onQ content. https://onq.queensu.ca/d2l/le/content/764101/viewContent/4551619/View