CISC 322
Assignment 1
Conceptual Architecture of Bitcoin Core
Sunday, February 19th, 2023

Authors:

Makayla McMullin	19mam51@queensu.ca
Aniket Mukherjee	aniketm7274@gmail.com
Daniel Dickson	d.dickson@queensu.ca
Maia Domingues	18mkd6@queensu.ca
Lucas Patoine	19lwp@queensu.ca

Table of Contents:

Abstract

In our report, we will be examining the high-level conceptual architecture of the software Bitcoin Core, a software used to facilitate Bitcoin transactions. Bitcoin Core was launched in January 2009 alongside the Bitcoin cryptocurrency (though the name “Bitcoin Core” for the software was only adopted in 2014) [1]. Since then, it has gone through many changes and updates, but the most recent of these, Version 24.0.1, will be our main focus in most sections.

In this report, we will examine the main architecture and architectural styles of the software. We will additionally look at its various subsystems and interfaces in order to fully understand how it functions, including the Network, Blockchain, and Wallet. We will evaluate previous versions of the software to look at these changes it has undergone in the last 14 years. We will also discuss how the software manages concurrent transactions via its memory pool, how the project was divided among developers through methods such as separating repositories and version control, and the use cases of making and mining a transaction that demonstrate how the software functions as it is used.

Our report will also include a reflection on challenges we faced, lessons we learned, and the limitations of our research. We will also include a glossary of naming conventions, and a data dictionary to help clear up some of the terms we use in the body of our report.

Introduction

Over the last decade, Bitcoin has become a disrupter in the financial world. As the most prominent cryptocurrency by market cap, Bitcoin has surely come a long way since its humble beginnings and dreams of a world where a digital, decentralized currency becomes mainstream. Bitcoin Core, the software that provides the foundation on which the Bitcoin cryptocurrency operates, was launched in early 2009 under the name “Bitcoin-Qt”, with the Bitcoin cryptocurrency launching alongside it. Without the underlying architecture and feature set that Bitcoin Core provides and has continued to expand upon as the Bitcoin blockchain grows larger and larger, Bitcoin could have never gained the same prominence that it has today.

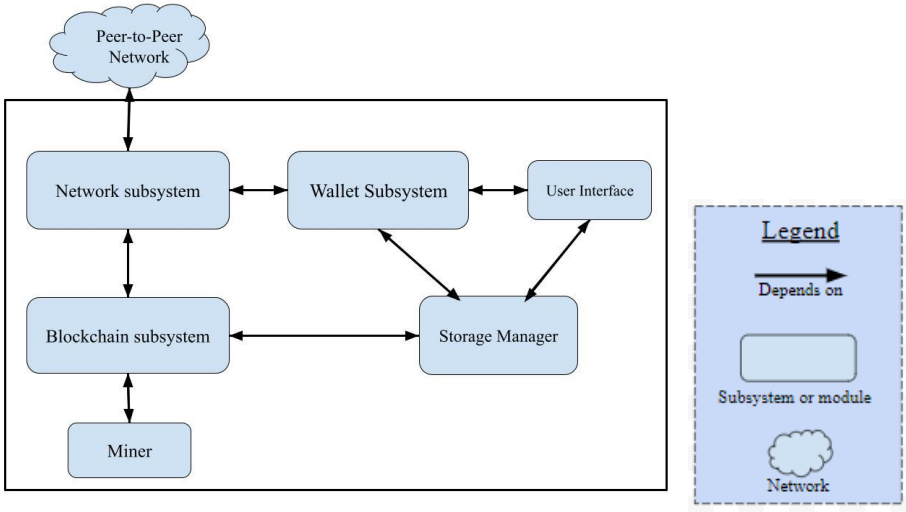
Bitcoin Core is open-source software, which means that anyone is able to access the source code and propose changes if they feel something should be integrated into the software. If enough people agree on a proposed change on the basis of a majority consensus, that change is then implemented into the next release of Bitcoin Core. Initially, Bitcoin Core only consisted of a command-line interface which let users manage their Bitcoin wallet, mine blocks on the blockchain, and process transactions. As more people gained interest in Bitcoin Core and its development, more changes were proposed to the software and integrated into the Core itself, strengthening the utility and feature set of the software. Ultimately, Bitcoin Core has grown to be highly functional and reliable and is the most popular implementation of the Bitcoin protocol.

As a peer-to-peer decentralized currency protocol, Bitcoin Core functions as the bridge between a user, or node, and the blockchain network that consists of all Bitcoin cryptocurrency transactions. Each node on the network plays a role in processing transactions across the entire

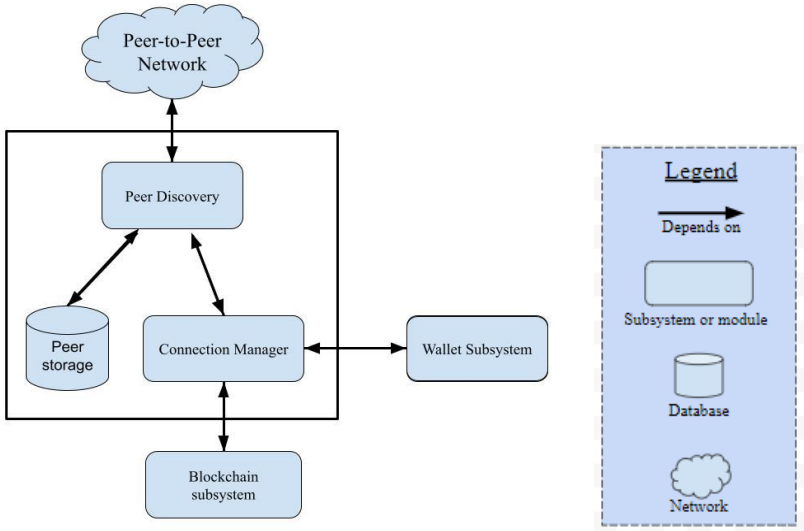
network, and therefore each instance of Bitcoin Core contributes to the running state of the network as a whole. Additionally, the Bitcoin blockchain is replicated across multiple nodes on the network to protect against a single point of failure in the system, and also to ensure the validity of the transactions on the network.

Architecture

Bitcoin Core uses an object-oriented architectural style that acts as a client and a server in the Bitcoin peer-to-peer network architecture style. The modularity of the system's architectural style supports future changes in the system because changes to one object in the system do not affect the others. Abstraction allows for new versions of the software to be developed easily. This ease is necessary since Bitcoin Core has hundreds of contributors, and due to the need for security updated versions must be released frequently. The following dependency diagram details the overall structure of the system.



Network Subsystem



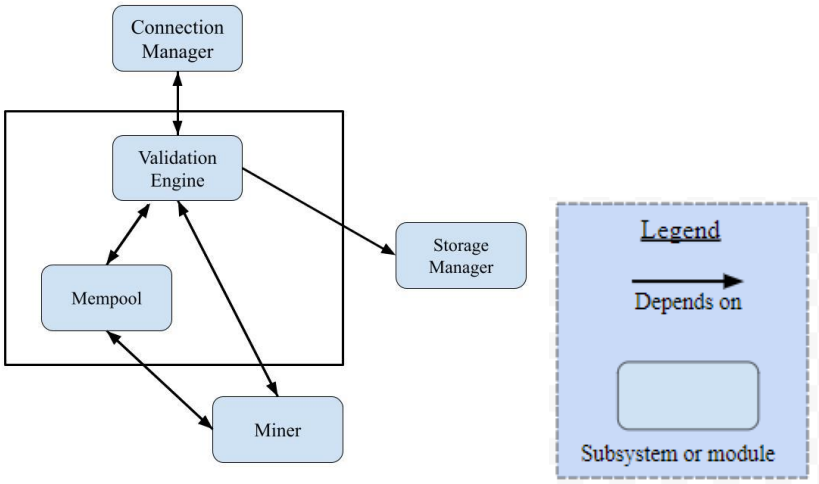
The Network Subsystem manages the communication between the Bitcoin Network and Bitcoin Core software system. The subsystem sets up the initial connection to the peer-to-peer network and maintains its connection through the exchange of peer, blockchain and transaction data.

To set up the initial connection to the Bitcoin Network, the Peer Discovery module must first find a pre-existing node in the network and establish a connection.Once Peer Discovery finds an established Bitcoin node, the two hosts will set up a TCP connection and the Bitcoin peer with send information about itself as well as a copy of the blockchain and a list of its neighbour peers. Since Bitcoin nodes are free to leave the network at any time it is important to make multiple connections between diverse nodes[2]. Therefore, whenever the Peer Discovery module connects to a new node it will request a list of their neighbours and attempt to reach out to any geographically, technologically, or categorically diverse nodes on that list.

To keep track of all of the system's peers, Peer Discovery collects information about nodes it connects with and sends that information to Peer Storage [3] (shown in the Network subsystem diagram above). This information includes the peer's IP address, the services the peer has access to, when the last connection with the peer was made, and how much data has been sent and received from the peer. Peer Discovery will periodically retrieve this information from Peer Storage to send or request new information from its neighbours.

The Connection Manager in the Network Subsystem is used to communicate transaction and blockchain information to the rest of the system. This involves communicating with the Blockchain Subsystem and the Wallet Subsystem. The Connection Manager sends any transactions that have yet to be mined and any new blocks it has received from its peers into the Blockchain Subsystem. The Blockchain Subsystem sends back any blocks it has completed and updates detailing if the subsystem is congested. The Connection Manager also communicates transaction information with the Wallet Subsystem. If the user makes a transaction the Wallet Subsystem sends the transaction to the Connection Manager to send to the Bitcoin Network. Furthermore, if a public key is requested by the user so that they can receive bitcoin, the Wallet Subsystem sends a copy of that key to the Connection Manager. As new transactions and blocks go through the Connection Manager, it scans them for any of the user's public keys. If a transaction to the user's public key is found the Connection Manager flags the transaction and sends it through the Blockchain Subsystem where it verifies that the transaction and sends it to the Storage Engine where it can then be viewed by the user interface.

Blockchain Subsystem



The Blockchain Subsystem is responsible for maintaining the system's local copy of the blockchain as well as contributing to the network blockchain. The subsystem accomplishes this by interacting with the Connection Manager in the Network Subsystem, the Storage Manager, and a Miner module.

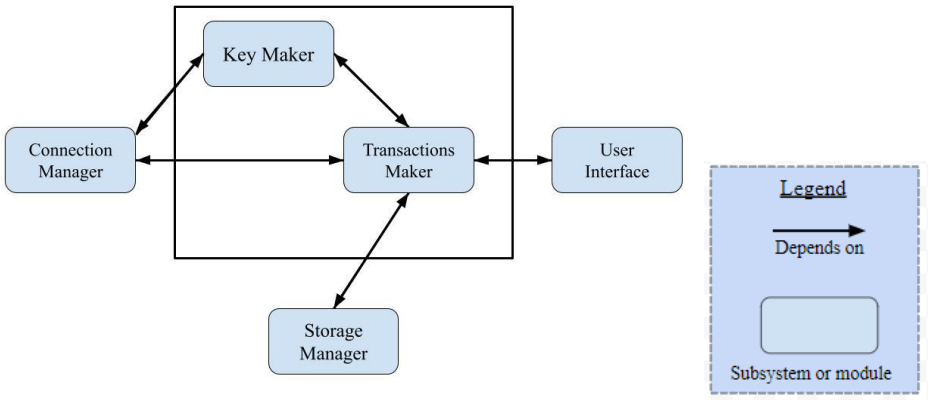
When an unverified transaction is propagated through the Bitcoin network and received by Network Subsystem, the Connection Manager sends the transaction to the Validation engine in the Blockchain Subsystem. The Validation engine checks that the transaction is "well formed, uses previously unspent outputs and contains sufficient transaction fees to be included in the next block" [4].

If the transaction is verified it is sent to the Mempool module. The Mempool stores all transactions in the network that are not yet in the blockchain and orders them based on which would be most desirable to mine. Transactions with higher transaction fees are placed at higher priority but other factors such as the difficulty of the work needed and the capacity of the miner are considered [5].

When the Miner is ready to start mining a transaction it will notify the Mempool and the Mempool will send a high-priority node to the Miner module. The Miner performs a cryptographic proof on the transaction and makes a new block. The block is sent to the Validation Engine where it is checked and if the block is correct one copy of it is sent to the Connection Manager subsequently in the Bitcoin Network and another copy is sent to the Storage Manager to be added to the local blockchain. If a new block is sent through the network while the Miner module is working the validation engine will send a message to the Miner asking if it is working on the same block. If the Miner was working on that block, it means that another miner completed work first and received the transaction fee. With no transaction fee to be collected, the Miner discards its work and selects a new transaction to work on[arch4]. Meanwhile, the validated block is sent to the Storage Manager so that it can be added to the local blockchain. See use case 2 for a sequence diagram describing this function.

It is important to note that Miner is a separate module that can be upgraded or downgraded without influencing the functionalities of the modules inside the Blockchain Subsystem. Therefore, the two parts support future changes to one another.

Wallet Subsystem



The Wallet Subsystem is responsible for making private and public keys, distributing these keys as needed, updating the storage system with references to received transactions, and making transactions to be sent into the network [6]. The subsystem performs these actions using two main modules: a Key Manager and a Transactions Maker.

The Key Maker creates private keys derived from a “seed” key. The Key Maker can then use the seed to determine if a key is derived from the seed. This process saves space because only the seed needs to be stored. Using the seeded wallet method is also beneficial if the user decided to switch to a different wallet system because only the seed needs to be copied over, not every key[7]. The Key Maker encrypts private keys into public keys. Then through hashing the Key Manager derives a Bitcoin Address. The address and keys can either be sent to the Transaction Maker so that bitcoin can be sent, or solely the Bitcoin Address can be sent to the user interface so that bitcoin can be requested. Either way, a copy of the Bitcoin Address is sent to the Connection Manager so any incoming transactions can be directed to storage.

The Transaction Maker interacts with the user interface, the Key Maker, the storage manager and finally the Connection Manager to compile and send transactions into the Bitcoin Network where they can be propagated and received.

If a user wishes to send bitcoin, a message containing the amount they want to send and the Bitcoin Address of the would-be recipient is sent through the user interface to the Transaction Maker. The Transaction Maker sends a message to the Storage Manager containing the amount of bitcoin the user wants to send. The Storage manager retrieves a list of unspent transaction outputs (UTXOs) that reference the user’s public key. In other words the parcels of bitcoin that the user owns. The Storage manager then picks one, or multiple, UTXOs that are equal to or greater than the amount of bitcoin the user wants to send. If the Storage Manager cannot find enough funds a cancel order is sent to the Transaction Maker and an error message is sent to the user interface, notifying the user that the transaction cannot take place. If there are enough funds the Storage Manager sends the UTXO(s) to the Transaction maker. The Transaction Maker then

requests new keys and an address from the Key Maker, the Key Maker makes the new keys and then sends them to the Transaction Maker. The Transaction Maker compiles an unsigned transaction and prompts the user interface to request user confirmation of the transaction. If the user confirms the user's private key is used to sign the transaction[8]. Once the transaction is signed it is sent From the Transaction Maker to the Connection Manager, then to the Peer Discovery and into the Bitcoin Network. The transaction is then received and broadcast by neighbouring nodes (which get to work on transaction verification) eventually making its way to the recipient's Bitcoin Node and subsequently wallet. See use case 1 for sequence diagram.

If a user wants to receive bitcoin the user interface will request the Key Maker to make new keys. The Key Maker will send a new Bitcoin address to the user interface where it can be sent to a sender who can use it to make a transaction and send it into the network so that the user can receive it.

Storage Engine Module

The Storage Manager module is responsible for directing the storage of the blockchain and the UTXOs referencing the user. The Storage Manager passes blocks to storage from the Blockchain Subsystem and vice versa. Furthermore, the Storage Manager is used by the user interface to retrieve the user's bitcoin balance and is used by the Wallet subsystem to retrieve UTXOs summing to a specific amount. Therefore the UTXOs need to be stored by the Storage manager in such a way that performing both functions is time efficient.

User Interface Module

Bitcoin Core provides a command line/API interface and a graphical user interface[9]. The user interface interacts with the Wallet Subsystem and the Storage Manager to allow the user to make transactions and check their bitcoin balance respectively. How the user interface interacts with the Wallet Subsystem to make a transaction is detailed in the Wallet Subsystem description. When the user wishes to check their bitcoin balance, the user interface sends a message to the Storage Manager which returns the balance to the user interface where it is displayed for the user.

Control and Data Flow Between Parts

The components of Bitcoin itself include: a decentralized peer to peer network, a public transaction ledger(more commonly referred to as the blockchain), a set of rules for independent transaction validation and currency issuance, and a proof of work algorithm, for reaching a decentralized consensus on the valid blockchain. Regarding the connection between all the global users of Bitcoin, it is a flat, decentralized peer to peer network where no node is given higher priority or granted greater access. In essence, the Bitcoin network is a collection of nodes running the Bitcoin p2p network protocol, while other Bitcoin related functionality like mining is hosted by external servers connected to Bitcoin through their own nodes. [4]

This is where Bitcoin Core comes into play. It essentially determines which blockchain contains valid transactions, and makes it so that Bitcoin Core users can only perform transactions on those valid blockchains. As a new wallet for bitcoin, it provides better validation and security for its users while making their transactions unlinkable to themselves(works hand in hand with

trustless blockchain validation), at the cost of more intensive computing. The way this is made possible is by the creation of different node types, all with their own functions, The wallet node(stores a users Bitcoin inventory), the blockchain node(stores a copy of the valid blockchain to append to later), the network routing node(responsible for connecting to the peer to peer network protocol that Bitcoin Operates on), and the miner node(responsible for solving the proof of work algorithm in exchange for more bitcoin). Depending on the context, these nodes can intertwine together in such a fashion that they seamlessly follow the larger, extended Bitcoin network, which involves various different protocols, depending on what the user is trying to do with Bitcoin Core(are they trying to engage in pool mining, or connect to Stratum network which facilitates communication between the main Bitcoin network, miners and mining pools that combine the hashing power of miners all across the globe?) In order for a node to participate, it first needs to find other nodes to connect to, to opt into the peer to peer network. To do this, we can query DNS with a number of DNS servers that provide a list of IP addresses that belong to Bitcoin nodes. From there, we can make even more connections to other nodes in the extended Bitcoin network, through automatic address propagation and discovery. [2]

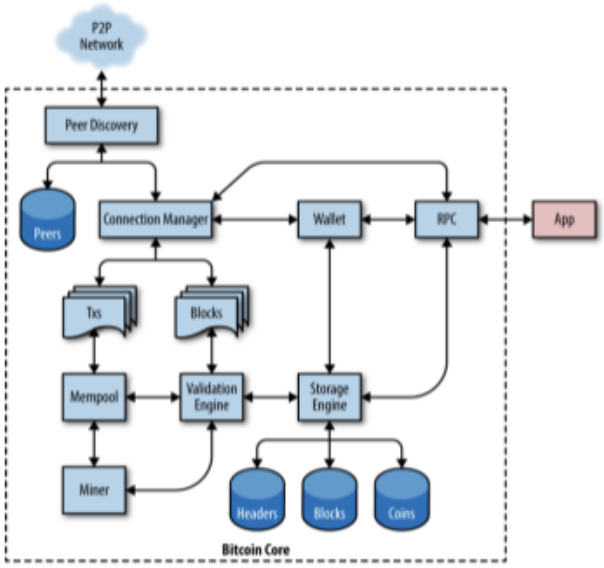
One such discoverable node is a full blockchain node, that maintains and updates a complete and up to date copy of all the transactions that occurred on that node. Full nodes can also authoritatively verify any transaction without using other nodes or sources of information to make sure it’s a valid transaction. Once verification is complete, it incorporates the verified network version of the full blockchain into the locally stored one, through the Bitcoin Core client. [3]One drawback of such a node structure is the sheer amount of space required to store a local copy of the blockchain, and all the data that comes with that. An alternative to full blockchain nodes is using simplified transaction verification nodes(SPV nodes for short)[2] which verify transactions by reference to their depth instead of their height. Put simply, SPV nodes verify the chain of all the blocks, but not their transactions, which makes them a less cumbersome option to blockchain verification. Unfortunately, this also makes them less secure as a whole, which means that to truly secure these nodes, special precautions must be taken.

One of the first things that a node will do when it manages to connect to peers is try to construct a full blockchain, by synchronizing with the network's version of the full blockchain. There is a comparison operation that occurs between the local blockchain and the one in the network whenever a node goes offline, which starts off by sending an inv response and then downloading the missing blocks.

The encryption of communication between the larger Bitcoin network and Bitcoin Core, as well as SPV nodes is typically done through the TOR network, which is also how the anonymization of transactions is prepared, as well as the enhanced security through the peer to peer connection of nodes. Bitcoin Core lets the user configure their settings accordingly to transport any bitcoin node and its data over the TOR network, allowing it to also remain unlinkable to them. Another security solution that is implemented by default(due to the nature of Bitcoin Core’s architecture) is peer to peer authentication and encryption, which serves to

prevent man in the middle attacks, as well as strengthen the resistance of bitcoin to governmental surveillance [2].

According to this architectural diagram of Bitcoin Core [2]:



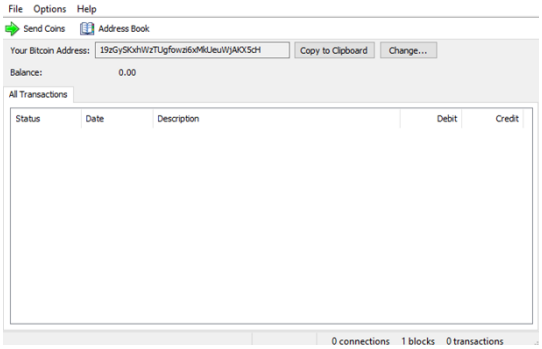
Once the app is started up, it initializes both the user's wallet and storage engine through remote procedure calls. This is to build up the distributed system that works seamlessly with Bitcoins usual architecture, and allow for regular, unmitigated use for the user. The storage engine stores both the coins in the wallet, as well as a copy of the verified blockchain, to append to when it gets updated. The validation engine is what performs the verification of the mempool that miners operate on, as well as the network version of the blockchain. The network version of the blockchain is obtained via the peer to peer connection that we've established via peer discovery on the peer to peer network. The miners get their information from the mempool, where it's a pool of blocks to work on and obtain bitcoin for the user using them.[2]

Evolution of a system

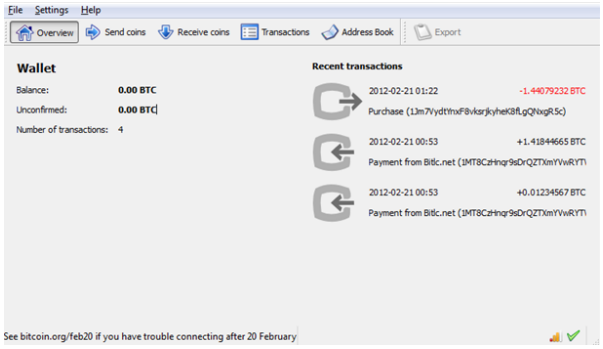
The initial framework for what would eventually become Bitcoin Core, along with the specifications of the peer-to-peer Bitcoin network itself was outlined in the whitepaper for Bitcoin, created by Satoshi Nakamoto: a pseudonym used by Bitcoin's creator. On September 1st, 2009, Satoshi Nakamoto uploaded version 0.1 of Bitcoin. Though simply called Bitcoin at the time, this software would later be renamed Bitcoin-QT when a QT graphical interface was implemented and would finally go by the name of Bitcoin Core starting at version 0.9.0. This rebranding was stated to have been done in order "to reduce confusion between Bitcoin-the-network and Bitcoin-the-software". [10] In comparison to its more recent versions, version 0.1 was relatively rudimentary, lacking many of the features that would later be implemented. While version 0.1 still acts as a Bitcoin wallet, allowing users to make transfers and mine bitcoins, as well as keep an address book of any recipients you may want to keep track of, it is pretty barebones otherwise.

At the time, Bitcoin was only available for Windows operating systems, and would remain that way until later that year when version 0.2 was released, adding support for Linux along with allowing for the use of multi-core processors in mining. The next key event that took place in the development of Bitcoin Core was Bitcoin version 0.3.9, where shortly after its release, Nakamoto left the project, completely disappearing off the internet. [11] Moving ahead a bit to the next major release, Bitcoin version 0.4.0 implemented optional wallet encryption that utilized a private key created by the user, and version 0.5.0 implemented the Qt user interface toolkit, hence renaming the software "Bitcoin-QT". Version 0.5.0 allowed for many GUI improvements, though implemented other changes as well, such as improving wallet encryption and adding RPC commands such as getmemorypool. [10]

Bitcoin Version 0.1.0:



Bitcoin Version 0.5.2:



Bitcoin-Qt 0.6.0, which was released March of 2012 added features such as a backup wallets, and QR codes for easily sharing addresses. 0.7.0, released later that year, replaced the aforementioned RPC command getmemory with getblocktemplate and submitblock. [10] Finally, in version 0.8.0, the last major release of Bitcoin-Qt before it would be rebranded as Bitcoin Core, Bitcoin-Qt was redesigned to use LevelDB to store most information (having priorly used Berkeley DB). This helped to significantly decrease blockchain synchronization time. [11]

In March of 2014, Bitcoin Core 0.9.0 was released, the first version under the name by which it is still known as today. What follows is a list of some of the major ways in which the software has changed and evolved in the updates since then:

Version Number	Release Date	Changes
0.9.0	19/03/2014	~ Software is now 64-bit with 32-bit no longer being supported. ~ Software renamed Bitcoin Core to avoid confusion with Bitcoin Network. ~ Required transaction fee is lowered.
0.10.0	16/02/2015	~ Switches to using headers-first synchronization, allowing for a faster block synchronization. ~ Introduction of a consensus library.
0.11.0	12/07/2015	~ Added block pruning to remove unnecessary raw data once it

		had been validated, however block pruning and running a wallet were not compatible at this time. ~ Support was added for CPUs implementing a big-endian architecture. ~ Decreased overall memory usage.
0.12.0	23/02/2016	~ Code for Bitcoin mining has been optimized such that less memory is used. ~ Added ability to reduce upload traffic. ~ Added hard limit on mempool size, configurable by users. ~ Possible to utilize block pruning and run a wallet at the same time.
0.12.1	15/04/2016	~ Several Bitcoin Improvement Proposals (BIP) are enforced through the deployment of a soft fork.
0.13.1	27/10/2016	~ Segregated witness (segwit) introduced as a soft fork with the necessary preparations for segwit being included in the release of 0.13.0 earlies that year. ~ segwit transactions, compared to normal transactions, allow for blocks containing them to hold more data. If each user were to use segwit transactions, the overall capacity of the Network would increase by around 70%.
0.14.0	08/03/2017	~ Significant increase in both validation speed and network propagation performance ~ Due to P2P refactoring which prioritizes concurrency and throughput, block fetching is much faster than in previous Versions. ~ Adds support for manual pruning, allowing the user to specify things such as the height to which a block is pruned.
0.16.0	26/02/2018	~ segwit now fully supported by wallet and user interfaces.

[12] [10]

Concurrency

In order for the Bitcoin Core network to run smoothly and securely, Bitcoin Core developers must overcome the challenge of handling concurrency. In this case, concurrency refers to the ability for the network to process multiple transactions at the same time. Because Bitcoin Core interacts with a decentralized peer-to-peer network, the software must be able to handle multiple concurrent transactions and transaction processing from multiple nodes spread throughout the network, and efficiently manage the flow of data.

One of the main ways that Bitcoin Core handles concurrent transactions is through the utilization of a memory pool or mempool. Each node on the network has its own mempool from which to draw potential new transactions from, and transactions are performed independently of another node on the network processing a different transaction, allowing for parallel processing

of transactions. Once transactions are validated and added to the mempool, the transactions are ready to be processed and confirmed to ultimately be added to the blockchain.

Before transactions are added to the mempool, they are validated by nodes across the Bitcoin network. To keep data consistent across the decentralized network and to further ensure security, Bitcoin Core asks each node on the network to validate the transaction concurrently. In this process of validation, each node will check to see if the transaction is properly signed according to Bitcoin Core protocol. If a node deems a transaction to be valid, it will add the transaction to its own mempool. Each node across the network will validate the transaction, and once a transaction has received validation from at least 51% of nodes on the network, it can be passed on and confirmed onto the blockchain. The process of receiving validation from the majority of nodes on the network is known as achieving network consensus. [13] Waiting for consensus before a transaction is deemed able to be added to the blockchain ensures that, in the case of two near-identical transactions taking place on a network at the same time where one is legitimate and one is either illegitimate or corrupted, the illegitimate/corrupted transaction will not make it onto the blockchain.

Once a transaction achieves consensus across the network, the transaction is ready to be confirmed and placed onto the blockchain. If a transaction is not confirmed within a certain amount of time, it will be taken out of the memory pool entirely, freeing up space for more transaction processing. To handle many concurrent transactions waiting to be confirmed and reduce the amount of transactions in a backlogged state, the order and priority in which transactions are chosen to be confirmed onto the blockchain is established. This is primarily achieved using transaction fees.

Transaction fees are determined by a Bitcoin miner and are applied to a transaction before being sent out to be confirmed. As the set fee increases, the likelihood that the transaction will be included in the next block added to the blockchain increases. In the event that multiple concurrent transactions are taking place on the network at the same time, Bitcoin Core will prioritize the transactions with higher fees, speeding up the confirmation process.

Divisions of Responsibilities among Developers

As Bitcoin essentially functions as its own currency system, one would expect that such a project will involve lots of time, effort, and people. This is evident in their GitHub account [14], which is associated with 16 publicly active and 4 publicly archived repositories, including one for their website, interface, GUI, documentation, and LevelDB (a system for data reading and writing that is used to keep track of data concerning Bitcoin transactions). Each repository also contains several different contributors. The amount of contributors and commits ensures that the project has enough resources to be developed efficiently and effectively, as well as enough stages of development and continuous improvement to the system to ensure that it remains polished and running smoothly. At the time of writing this report, some of the newest commits were mere hours old.

Separation of Project Components

As previously stated, there are 20 available repositories associated with the Bitcoin Core GitHub account. Some of these repositories contain over 35 000 commits, while others have less than 100, but most are over 1 000 each. These commits showcase all of the effort and continuous improvement the project as a whole has gone through, and continues to go through. The separation of various components is useful for keeping better track of each unique element, as organization would be crucial for a project of this size with a large number of parts that may constantly need improvement and regular maintenance. A detailed analysis of the top 100 contributors from each repository (or all contributors, if there were less than 100 in total) showed that 61.7% of unique contributors only made contributions to one of these repositories, and the average unique repositories contributed to per user was 1.84. Ensuring that some individuals developing this project are familiar with many of its components (those who contributed to several repositories) ensures that the different parts can be connected in the right ways. Meanwhile, having many individuals only focusing on one part allows them to specialize and become especially familiar with these parts, ensuring that if maintenance is needed, there are people who have specific expertise and can more easily make the necessary changes. This also may prevent overworking of the developers; given that they only have one part to focus on and understand instead of several, it may be easier and quicker for them to build the system without feeling overworked or lost.

Version Control and Archiving

As is apparent from both their use of GitHub [14] and their website homepage [15], the developers release numbered versions of the software for public use. A system like this helps developers keep track of when features were released, when bugs were fixed, and when enhancements were added, among other important details. Similarly, each website post about a new version release also contains release notes, which describe in detail the changes that were made between the current version and its predecessor. Each change summary in the list also links to the corresponding GitHub pull request, which shows the initial issue, which developer(s) implemented the change, precisely when the change was implemented, and each step along the way of implementation (including commits, comments, status changes, etc.). Their numbers seem to use a major.minor system, where a major release involves a large number of "notable changes" that greatly impact the software's functionality or support, whereas a minor release involves a smaller number of bug fixes or minor system changes. It appears that previous versions of Bitcoin Core used a 0.major.minor system, until version 0.21.2 upgraded to version 22.0. It is unknown why this was changed. Archiving older versions also ensures that the developers can look at and potentially rollback to previous versions of some components should they find major issues or vulnerabilities with a new version.

External Interfaces

The Bitcoin Core system interacts with two external interfaces: the Bitcoin Network and the user interface. All input and output are sent through these interfaces.

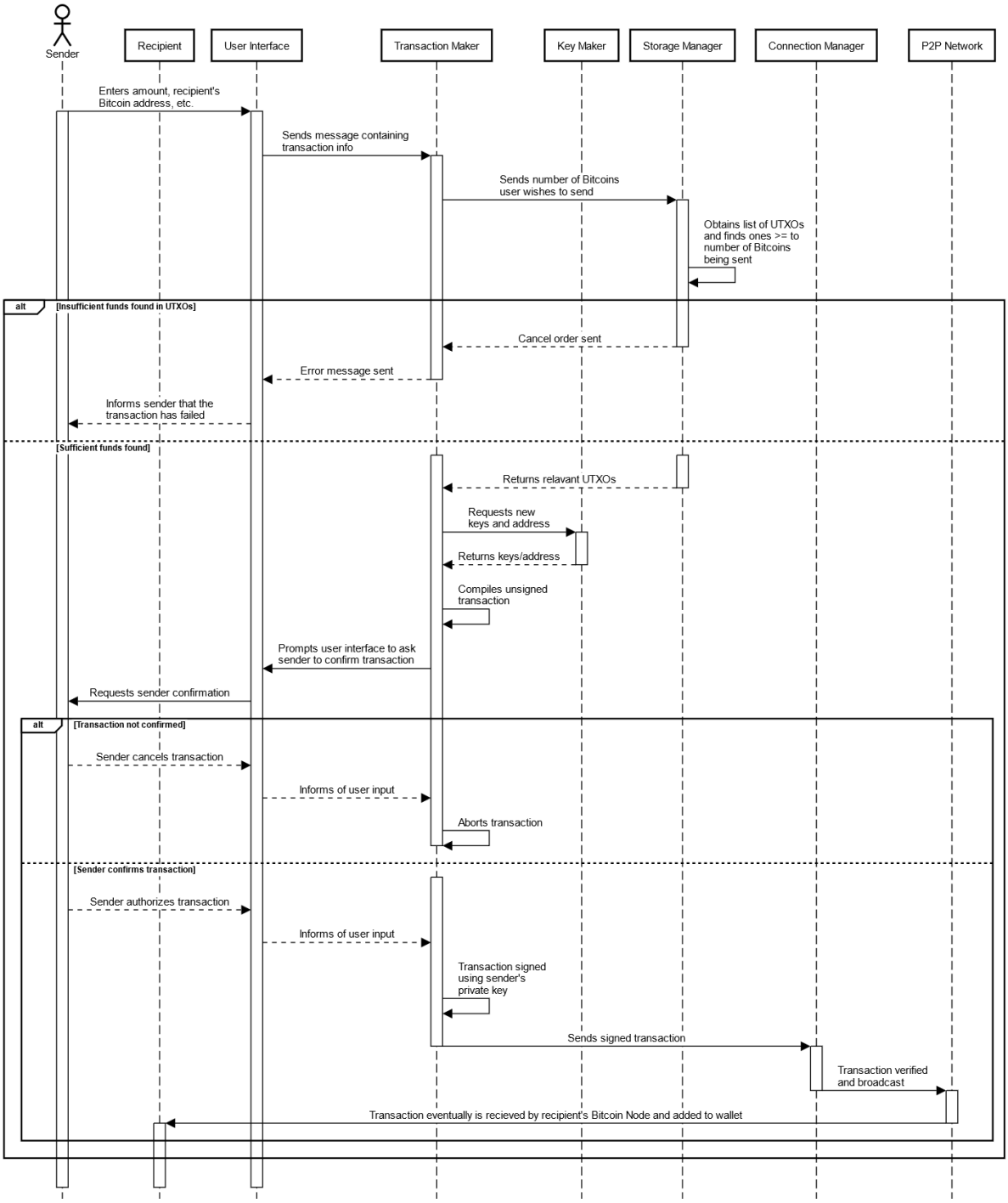
The Bitcoin Network interface is how the system connects to its Bitcoin peers. Transactions, blocks and peer information are sent from the Bitcoin Network to the Bitcoin Core system and vice versa.

The user interface manages input and output regarding user transactions and bitcoin balance. There are four main functions the user interface fulfills: sending bitcoin, Bitcoin address requests, checking a user's bitcoin balance and viewing a user's past transactions. When the user wants to send bitcoin, they input the amount of bitcoin and the Bitcoin address of the recipient into the user interface. The Bitcoin Core system will return whether or not the transaction went through to the user interface. When the user wants money sent to them, they will request a Bitcoin address through the user interface.The Bitcoin Core system will then respond with the address. When the user wants to check their balance or view past transactions, the user interface sends a message request to the Bitcoin Core system, and the system responds with the information.

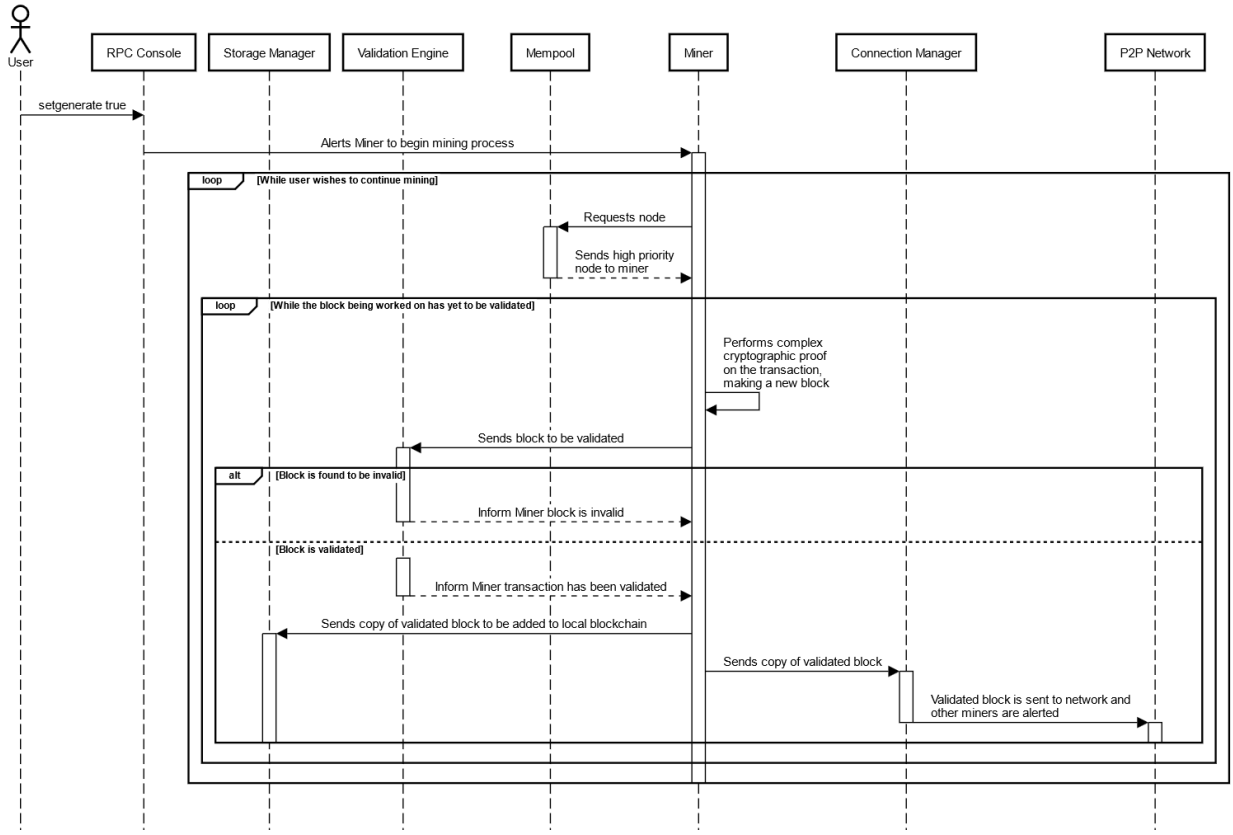
Use cases

Going off of the conceptual architecture discussed thus far, what follows are two sequence diagrams detailing how different components communicate with each other for two distinct use cases: making a transaction, and mining Bitcoin.

Use Case 1: Making a transaction



Use Case 2: Mining Bitcoin



Conclusion

Ultimately, we have observed that Bitcoin Core uses an object-oriented architectural style that acts as a client-server style when connected with Bitcoin's peer-to-peer structure. This is an effective architectural style due to the large number of contributors and the constant need for updated security due to the nature of the system. We examined the various subsystems of Bitcoin Core, including the Network (communicates between Bitcoin Network and Core), Blockchain (manages blockchain information), and Wallet (creates and distributes keys and stores transaction information). We've examined all the updates to the system over time, and how developers have divided these updates and systems in an effort to make the project more manageable for their team. We also examined the Network and User interfaces and their functions.

Overall, Bitcoin Core is a revolutionary technology that is having a great impact on how our world perceives currency. Its peer-to-peer network system is very useful for a system that relies so much on human interaction, while its object-oriented components make it very useful for developers to continuously make updates and improvements as necessary. Furthermore, its decentralization makes it much easier for users to be independent in and feel safer with their interactions. Bitcoin Core has commenced a new era of e-commerce and currency exchange, and our group looks forward to getting to further interact with and learn more about the system.

Limitations and Lessons Learned

Due to the amount of assessments and deadlines from other courses recently, it has been very difficult to coordinate meeting times without groups and set aside time to work on the project. Additionally, we had an issue resulting in the restructuring of our group and redistribution of some responsibilities very last-minute. Due to this, we may have rushed through some aspects of our report and presentation, though our group members did the best they could to accommodate our schedules accordingly to ensure enough time to complete our work as carefully as possible. Additionally, we found that we had allocated too much of our time to researching and preparing for our parts, leaving us with less time to actually summarize the information and use our sources within our report. Going forward, we will distribute time more evenly to ensure that we have the right amount of time for both actions. Furthermore, we may have initially misinterpreted the complexity of our software. While knowing Bitcoin Core is a very complex system, we were initially underestimating the amount of subprocesses and functions we would have to analyze for our project. For the next assignments, we will also consider time necessary to completely understand how everything works (assignment instructions and writing) before we begin to contribute our respective parts.

Our research was mildly limited by how rapidly Bitcoin Core is developing. Even in more recent days, their GitHub was still being modified within hours of us accessing it for research. Bitcoin Core is constantly and actively expanding and being improved, and thus, some aspects of our report may grow outdated in the near future as a result. Additionally, some data about earlier versions of Bitcoin Core proved more difficult to access when researching, so a lot of our data is based on the current system and we were unable to use some older versions for architectural analysis to assist us with our understanding of how the modern software functions.

Naming Conventions

UTXO (Unspent Transaction Output): technical term for an amount of bitcoin that in a transaction

API(Application Programming Interface): software that dictates the communication between two applications.

DNS (Domain Name System): a service that translates domain names into IP addresses.

SPV nodes: Short for simplified payment verification nodes. These note can verify payments without needing to download the entire blockchain.

Mempool: a storage space for transactions not yet added to the blockchain

Miners: computers that add blocks to the blockchain by performing algorithmic calculations.

GUI (Graphical User Interface): what users use to interact with computer software.

Data Dictionary

Bitcoin vs bitcoin: Capital is used when referring to the technology, concept or network. The uncapitalized version refers to the unit of cryptocurrency.

Peer-to-peer architecture style: a type of computer network architecture in which there is no division or distinction in abilities amidst various workstations or nodes on a network.
Object-oriented architecture style: maps the application to real world objects for making it more understandable. Provides reusability through abstraction and polymorphism.

References

[1] Wikipedia Contributors, “Bitcoin,” *Wikipedia*, 17-Feb-2023. [Online]. Available: https://en.wikipedia.org/wiki/Bitcoin.

[2] A. M. Antonopoulos, ”The Bitcoin Network”, in *Mastering Bitcoin: Unlocking digital cryptocurrencies*. Sebastopol, CA: O'Reilly, 2015.

[3] A. M. Antonopoulos, ”Bitcoin Core: The Reference Implementation”, in *Mastering Bitcoin: Unlocking digital cryptocurrencies*. Sebastopol, CA: O'Reilly, 2015

[4] A. M. Antonopoulos, ”How Bitcoin Works”, in *Mastering Bitcoin: Unlocking digital cryptocurrencies*. Sebastopol, CA: O'Reilly, 2015

[5] A. M. Antonopoulos, ”The Blockchain”, in *Mastering Bitcoin: Unlocking digital cryptocurrencies*. Sebastopol, CA: O'Reilly, 2015

[6] “Wallets,” *Bitcoin*. [Online]. Available: https://developer.bitcoin.org/devguide/wallets.html.

[7] A. M. Antonopoulos, ”Keys, Addresses”, in *Mastering Bitcoin: Unlocking digital cryptocurrencies*. Sebastopol, CA: O'Reilly, 2015

[8] A. M. Antonopoulos, ”Transactions”, in *Mastering Bitcoin: Unlocking digital cryptocurrencies*. Sebastopol, CA: O'Reilly, 2015

[9] “Bitcoin Core's user interface,” *User Interface - Bitcoin Core Features*. [Online]. Available: https://bitcoin.org/en/bitcoin-core/features/user-interface.

[10] “Bitcoin Core Version History” Bitcoin. 2023 [Online]. Available: https://bitcoin.org/en/version-history

[11] Xu, H. “Bitcoin Core” Encyclopedia. 2022, November 09 [Online]. Available: https://encyclopedia.pub/entry/33603

[12] “Bitcoin Core Releases” Github. 2023 [Online]. Available: https://github.com/bitcoin/bitcoin/releases

[13] “What is Blockchain? From the Byzantine Generals Problem to Consensus,” *crypto.com*. 2022 [Online]. Available: https://crypto.com/university/what-is-blockchain-consensus .

[14] “Bitcoin Core Repositories,” GitHub. 2023 [Online]. Available: https://github.com/orgs/bitcoin-core/repositories

[15] Bitcoin Core, “Bitcoin,” *Bitcoin Core*, n.d.. [Online]. Available: https://bitcoincore.org/.