CISC 322/326
Assignment 1: Conceptual Architecture of Bitcoin Core
February 17, 2023

Manny Cassar	m.cassar@queensu.ca
Sawyer Proud	19sqp@queensu.ca
Cameron Krupa	19cmer@queensu.ca
Marcus Tantakoun	20mt1@queensu.ca
Duncan Scanga	19dms7@queensu.ca
Eric Jin	e.jin@queensu.ca

Table of Contents

Abstract

In this report, our team, Big Generational Wealth, analyzes the high-level conceptual architecture of Bitcoin Core, an open-source software platform that serves as the reference implementation for the digital currency Bitcoin. First released in 2009, Bitcoin Core is foremost responsible for managing decentralized digital currency that operates without the need of financial system and government authority. This discussion covers the conceptual framework derived by our team and the reasoning for the decisions behind these derivations, along with the descriptions of individual subsystems and their interactions amongst each other identified in the resulting architecture. Additionally, we provide two fundamental use cases.

As stated by the developers of Bitcoin Core, the conceptual architecture aligns itself as a network of peer-to-peer transactions – a key factor in the decentralized nature of Bitcoin that makes the network so secure and resilient. It includes eight components or subsystems, namely: RPC, Wallet, Miner, Storage Engine, Validation Engine, Mempool, Connection Manager, and Peer Discovery. Detailed in this report, each component and its related processes provide critical functionality and interactions within the network system. Further, sequence diagrams describing the use cases of (i) completing an online transaction purchase, and (ii) an owner checking their wallet balance have been used to display the control and data flows within the derived architecture. Bitcoin Core's evolution, its concurrency, how participating developers divide their responsibilities within the context of this conceptual architecture, as well as external interfaces are also discussed. The report's conclusion summarizes all major findings and suggests possible directions for future work. Beyond the scope of Bitcoin Core, our report also highlights the limitations encountered by the team and lessons learned during the research and writing process.

Introduction

Decentralized currencies like Bitcoin demonstrate a major shift in the way we think about money and financial systems. Their rise not only represents technological advancement, but also a political and social statement about the importance of individual sovereignty and control in our financial systems. In 2008, Bitcoin was created by an unknown individual or group using the pseudonym Satoshi Nakamoto. Satoshi introduced the concept of Bitcoin in a whitepaper describing a digital, decentralized currency that could allow for secure and direct transfers of value without the need for intermediaries. In 2009, Satoshi released the first version of Bitcoin Core which provided the foundational framework infrastructure for the Bitcoin network.

This report describes Bitcoin Core's conceptual architecture, and its findings are predominantly derived from documentation that can be found in the official GitHub repository for the project. Available documentation details Bitcoin Core's different components and their interactions, the software's evolution and version changes, and how to use the platform and contribute to its development. From this, our team concludes that Bitcoin Core implements a "Peer-to-Peer" (P2P) software architecture style to facilitate communication between individual participants rather than through a central server or authority. This allows for a more distributed and

decentralized system which eliminates the possibility of a single point of control or failure. Each node in the network acts as both a client and a server which uses P2P communication to transmit transaction data and updates to the blockchain ledger. This ensures equal distribution of responsibilities and the security and stability of the network even if individual nodes are compromised or unavailable. In the Bitcoin Core system, we concluded that these refer to the eight components: RPC, Wallet, Miner, Storage Engine, Validation Engine, Mempool, Connection Manager, and Peer Discovery.

After outlining our team's derivation process and rationale for the proposed architecture, we analyze each component by describing their respective functions and interactions with other components, including the control and data flows among them. This provides a clear understanding of how these subsystems fit within the proposed architecture. We then explore the evolution of the Bitcoin Core system, the impact of the conceptual architecture on the allocation of responsibilities for participating developers, and the system's concurrency and external interfaces. To support our findings of a P2P structure, context is provided by illustrating two use cases pertinent for the digital transaction network platform: (i) completing an online transaction purchase; and (ii) owner checking wallet Bitcoin balance.

The report's conclusion provides an overview of our main findings alongside suggestions of direction for future endeavour. Additionally, our team has documented notable limitations and learned valuable lessons throughout the process. By constructing and examining the conceptual architecture of Bitcoin Core, our team has gained a comprehensive understanding of the system's structure, its functional requirements, and key relationships between its subsystems/components.

Architectural Overview

The peer-to-peer architectural style of Bitcoin core allows cash transfers electronically from one party to another. This architectural style implements a network of nodes which act as both a client and a server. This style is integral to decentralization and eliminates double spending while fully validating transactions. Value transfer is done through Lightweight wallets that follow the rules of blockchain validity and are highly scalable since addition of new nodes is easy. Resource sharing ensures only valid bitcoins are allowed on the network and is used for access control, enabling users to securely access funds through any client. Since only trusted miners are permitted to continue the blockchain by verifying a mined block's validity, security can only be compromised if the majority of miners are compromised. Every transaction is validated through a node which supports decentralization and prevents attackers from stealing. Bitcoin Core implements three different ways to validate a transaction. Built-in wallet's GUI or CLI/API interface can be used as a trusted peer for certain lightweight wallets. In either case, privacy is guaranteed by obfuscating the user's identity for each transaction placed. For example, a private transaction is 8a9djehd0e transacts 100 bitcoins to 9gsj3753hd. In this way, received transactions are secured with information-theoretic privacy. Another benefit of this architectural style is the low cost of implementation since workstations can share hardware and underlying software is already included

in the OS. Further, the network can adapt to changes which makes it flexible. In conclusion, the peer-to-peer architecture style provides a functional architecture that is well suited for Bitcoin core.

Alternative style

Alternatively, Bitcoin Core could be implemented using an object oriented architectural style since it is able to map applications to objects, making the programming style more understandable. There are many advantages that an object oriented architectural style possesses. Primarily, its object and class-based structure make the system "understandable, testable, maintainable, extensible, robust, and reusable" (Unknown, 2017). First, an object allows instances of a class to have interchangeable properties which would be useful in the implementation of Bitcoin Core since transactions on object wallets would be able to edit Bitcoin balances. Formulation of classes and polymorphism would allow sharing of an external interface to the blockchain structure enabling the exchange of Bitcoins between users. Encapsulation binds transaction data, securing the flow of value between identified wallets. Inheritance would allow Bitcoin Core to reuse repetitive code responsible for allowing a user to buy and sell. Message passing facilitates communication between users, enabling transactions to operate on the network. Additionally, association is imperative since fiat currencies must be tradeable for Bitcoins to initially access the network if you are not a miner. Object modeling provides relationships for objects and is necessary for classification of wallets and Bitcoins. Dynamic modeling attaches a response from a transaction to Bitcoin objects, making it possible to have a method to share Bitcoins through exchange. The functional model determines a transaction's function and allows distinction between valid and invalid Bitcoins. A conceptual, high-level design permits Bitcoins to be transitioned through the classes made possible through an interface. The detailed design will share detailed transaction information with the user once a purchase is made. In an object-oriented architecture, security is based on "the risk factors and context which would be implemented to be sustainable to Bitcoin core's system" (Saini, 2013). In sum, object-oriented architecture provides a safe way for Bitcoin core's system to transact Bitcoins in an efficient manner.

Components

Bitcoin Network Node

All nodes include four key abstractions: wallet, miner, storage/full blockchain database, and network routing. The routing function participates in the network and can include other functionality. All nodes validate and propagate transactions (Txs) and blocks, and discover and maintain connections to peers. The detailed and expanded implementation and functionality of these partitions are discussed below.

Wallets

Wallets act as a storage device for each user that can securely manage private keys, generate and sign transactions, and keep track of the user's Bitcoin balance. This component covers

two different kinds of wallets: Wallet Programs and Wallet Files. First, Wallet Files hold the collection of private keys either digitally using a file on a desktop, or physically by writing the keys down on a piece of paper. Second, Wallet Programs are software applications that allow the user to manipulate the Wallet File. Wallet Programs can use two types of key creation: deterministic or nondeterministic. Nondeterministic Wallets create keys that are independent from each other and are generated by randomizing letters and numbers. This type of key creation isn't used by many Wallet Programs since the requirement to back up each key individually makes them difficult to manage. Deterministic Wallets create a randomly generated seed and then use a one-way hash function that mixes index numbers with the seed to derive keys. This eases the process of managing multiple keys as you are able to derive all keys from a parent seed. Hierarchical Deterministic Wallets are a more advanced form of Deterministic Wallet which uses the BIP-32 standard to create a tree structure for the keys. This structure is created by one parent key that creates multiple child keys which then in turn can make grandchild keys (this can go on indefinitely). Classification of different branches allows better organization of the keys. Signing-Only Wallets and Networked Wallets work in conjunction to allow the user to both generate and sign transactions. First, Network Wallets are responsible for managing Bitcoin addresses and interacting with the peer-to-peer network. When a user initiates a transaction on a networked wallet, Signing-Only Wallets are signaled to make the user review the generated, unsigned transaction and sign it. The two most common variants to Singing Wallets are Offline Wallets and Hardware Wallets. Offline Wallets heighten the security from attackers by disconnecting from the network and working in tandem with a Networked Wallet that does not store any private key. These allow the user to monitor the balance and transaction history of a particular Bitcoin address without being able to spend the funds associated with that address. A Hardware Wallet is a device that eliminates the vulnerabilities of an OS by only being able to be used as a Signing-Only Wallet. To distribute keys exclusively, a Distributing-Only Wallet Program can be used. However, the simplest and most commonly used Wallet Program is the Full-service Wallet which performs all features mentioned above.

Miner

The Miner component is how users validate transactions and create new blocks on the Bitcoin blockchain, generating Bitcoins for themselves in return. When a transaction is done on the network, miners validate transactions by checking the corresponding digital signature, verifying that the sender has sufficient funds, and confirming that the block's hash meets the required proof-of-work criteria. If the transaction clears, it is added to the transaction history of the block chain. A new block is created every 10 minutes by talking every validated transaction that has happened since the last block that was added to the chain. However, to validate these blocks, miners compete against each other by solving complex puzzles using their computational power. The incentive for helping is ensured through transaction fees and a reward for a successful block mined. Miners work in one of two ways: solo mining or pool mining. First, solo mining is done by one miner where they get a new transaction by issuing a pull RPC command and competing against other users to validate the block. If successful, this solo miner receives all the block reward. However, solo mining is resource intensive and isn't viable for all hardware setups. Alternatively,

pool mining is done by multiple users, where a group of users combine their computational power and each user receives a reward dependent on their contribution to the block validation.

Storage engine

The Storage engine is a component of Bitcoin core that assures reliable and effective data storage and retrieval within the blockchain database. This component manages the types of data related to the Bitcoin Core such as the UTXO and transaction data within blocks and related information on the blockchain database. This is accomplished by updating the UTXO set to reflect changes in the block and managing the transaction data associated with it. Subsequently, this new block is mined by a node to update the blockchain database.

Validation Engine

The Validation Engine (VE) component of the Bitcoin Core software architecture is responsible for processing blocks and verifying transactions and is critical to the integrity and security of the Bitcoin network. When a new block is received by a node on the network, the VE checks if the block is valid by verifying its proof-of-work and the correctness of the transactions included in it. The engine also checks that the block's hash meets the target difficulty and that it does not violate any of the consensus rules of the Bitcoin network. If the block passes all of these checks, it is added to the local blockchain via the storage engine and broadcast to other nodes on the network. Similarly, when a new transaction is received, the VE verifies that it is well-formed, correctly signed, and does not attempt to spend more bitcoins than are available in the inputs. If the transaction is valid, it is then included in the node's Mempool and is considered for inclusion in future blocks. The VE and Miner components work in tandem to ensure that the Bitcoin network maintains security and stability. Ensuring this requires that miners only include valid transactions in the block they create by obtaining the necessary information about valid transactions from the VE. In turn, the VE provides the Miner with information about which transactions are currently in the Mempool and which have already been included in the blockchain. Additionally, the VE in Bitcoin Core has a direct connection with the storage engine, as both components work together to ensure the integrity and security of the blockchain. As the VE verifies the validity of transactions and blocks, it updates the blockchain state maintained by the storage engine by adding new transactions and blocks to the database or removing invalidated ones. The storage engine, in turn, provides fast access to the blockchain data, allowing the VE to efficiently retrieve and verify previously validated transactions and blocks.

Mempool

In essence, the Mempool keeps track of a "pool" of transactions that have not yet been included in a block but have been broadcasted to the network. The Bitcoin Core system Mempool is closely connected to a number of components. It has a critical relationship with the VE since it is in charge of making sure that only legitimate transactions are added to the Mempool. First, the mining component is connected to the Mempool, enabling retrieval of transactions to facilitate the creation of new blocks. A user interface for broadcasting and creating transactions to the network

is included to maintain ease of use. After a transaction is broadcasted to the network, every node that receives this new transaction has it added to its Mempool. It then stands by for a miner component to incorporate it into a block. First though, a node's validation engine must validate a transaction by checking to see if it abides by the network's consensus rules. If the transaction is legitimate, it is added to the node's Mempool and is held there until the miner decides to include it.

Connection Manager

The Connection Manager (CM) is an important component of the Bitcoin Core client. It is responsible for managing connections to other nodes in the Bitcoin network. A simple-payment-verification (SPV) client connects to Bitcoin nodes and requests access to transaction information. Lightweight clients interact directly with the network without an intermediary. The transaction is transmitted via the peer-to-peer protocol, quickly propagating across well-connected nodes in the network which see the receiver's address for the first time. The CM maintains a list of candidate nodes that the client can potentially connect to and periodically attempts to connect to them. When a connection is established, the CM will negotiate the version of the Bitcoin protocol that will be used for the connection and exchange information about the client and peer's respective capabilities. The CM also manages the number of connections a client has to other nodes. By limiting the number of incoming and outgoing connections, it ensures that the client does not get overloaded with traffic. Additionally, the CM also maintains the quality of the connections the client has with other nodes by sending ping messages to connected nodes to measure their latency and responsiveness; high latency or unresponsive nodes will be disconnected. Overall, the CM is a crucial component of the Bitcoin Core since it maintains the stability and reliability of the Bitcoin network by ensuring the client has a stable connection to the network and can effectively and efficiently communicate with other nodes.

Peer Discovery

The Bitcoin Core client uses a peer-to-peer network to relay transactions and blocks between nodes on the network. The Peer Discovery component facilitates this communication by discovering and connecting to other nodes. When a new node starts up, it will connect to a set of pre-configured seed nodes which are hardcoded into the client software. These seed nodes are operated by trusted members of the Bitcoin community and serve as initial entry points into the network. The new node will send a "version" message to the seed nodes which contains information about the client's software version, network protocol version, and other metadata. Once the seed nodes receive the version message, they will respond with their own version message which includes information about their software and network capabilities. If the new node's software is compatible with the seed node, they will establish a connection and exchange a list of other known nodes in the network. The new node will then attempt to connect to each of the known nodes on the list, sending a "version" message to each one. If a node responds with a "verack" message indicating that it has accepted the new node's version, the two nodes will exchange their full list of known nodes in the network. The Peer Discovery component also includes a number of mechanisms to maintain and manage connections to other nodes. For

example, nodes will periodically send "ping" messages to each other to verify that they are still connected and responsive. If a node is unresponsive, it will be removed from the list of known nodes.

RPC

The Bitcoin Core client includes a Ruby library wrapper for the JSON-RPC (Remote Procedure Call) API which provides developers a way to interact with a Bitcoin Core node and perform various operations related to the Bitcoin network. The RPC API is divided into different categories: Blockchain, Control, Generating, Mining, Network, Rawtransactions, Util, and Wallet RPCs. Using the JSON format, the RPC interface supplies information on the Bitcoin Core client status and provides status reports on different modules, making it easy for developers to interpret and understand data. Additionally, a command-line helper called bitcoin-cli allows users to experiment with the capabilities of the API. One of the main functionalities of the Bitcoin Core RPC is to provide users with information on the Bitcoin Core client status including the version numbers for the software client and protocol, the current number of connections, and various information about the network and client settings. Important commands for this include getblockchaininfo, getmempoolinfo, getnetworkinfo, and getwalletinfo. The RPC interface also allows users to explore and decode transactions by displaying all the components of a transaction, such as inputs and outputs which can be tracked by their transaction ID (txid). Transaction commands such as getrawtransaction and decoderawtransaction are used for this purpose. Finally, the RPC interface provides users with the ability to explore blocks which can be referenced by block height or block hash using commands such as getblock and getblockhash. This makes it easy for developers to further examine the blockchain and its contents. In conclusion, the Bitcoin Core RPC API is a powerful tool for developers to interact with the Bitcoin network and its nodes, providing a wide range of functionalities that can be easily interpreted by all programming languages.

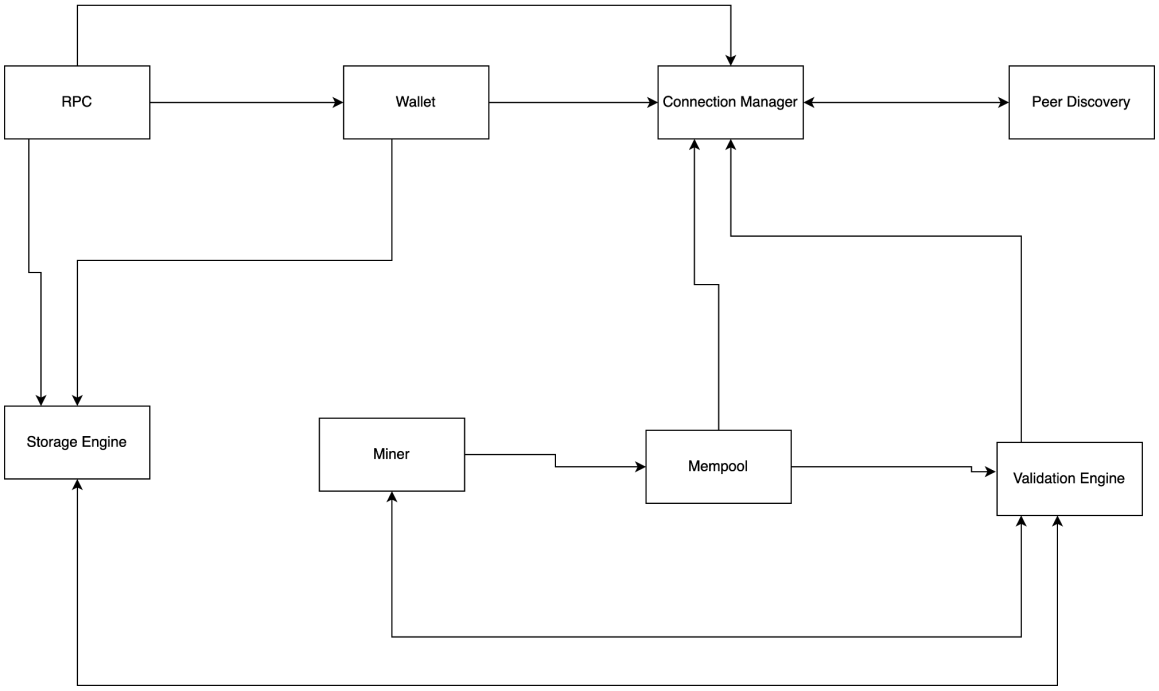

Figure 1: Box-and-Arrows Diagram of the Components

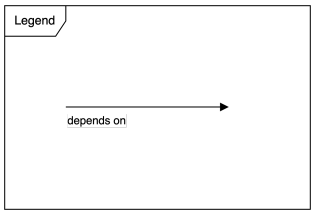

Figure 2: Legend for Box-and-Arrows Diagram

Use Cases

Use Case 1: A customer wants to complete a purchase (transaction) using bitcoin

The first use case assumes that the user (a customer) already has an account, and all necessary software installed on their local machine. This use case is meant to outline the process of a normal, successful transaction. The process begins with the user scanning a QR which contains the necessary information about the requested payment. The App component, which is how the user interacts with the system, will connect to the RPC component, which will first establish connection to the network of peers by way of the Connection Manager. Once connection is established, the RPC component can begin the procedures to complete the transaction. It first determines the transaction details and then asks the user if they want to confirm the transaction amount. If the user declines the transaction amount or details, the process will be halted. Once the user has agreed to complete the transaction, the RPC will send out a procedure to send the

transaction. To accomplish this, the Wallet component will be required to ensure that there is a UTXO of the correct value, along with determining the implicit fee paid to the Miner(s). Once a block is found, the RPC will then communicate with the Connection Manager component to process this selected block. Although not shown in the diagram to avoid confusion, if a block cannot be found or the account requested is more than the user's current balance, an error will be reported to the user and the process will be halted. The Connection Manager will communicate with the Peer Discovery component to find peers as they will be required to validate and complete the transaction. The Connection Manager will then work with the Validation Engine to ensure that the transaction is valid. This process requires Miner components to send the data across the network and ensure there are no violations. Finally, the Mempool is used by the Miner component to ensure valid transactions are being combined into blocks. If the transaction is deemed invalid, an error message will be reported to the user and the process will be halted with no change in either the sender or receiver's balances. Once the transaction has been completed and the transaction has been recorded across the network, the Wallet component will need to update its balance for both the sender and the receiver. Finally, the process of this transaction will be completed.

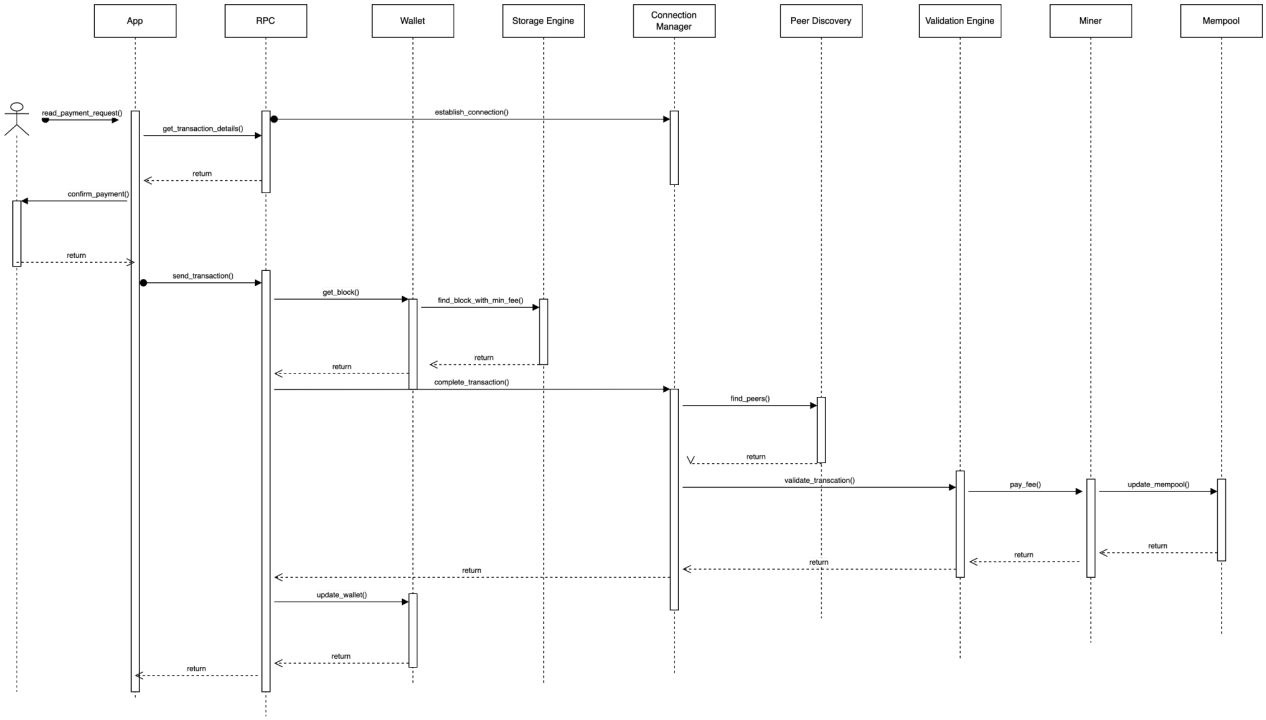

Figure 2: Sequence Diagram of Completing a Transaction

Use Case 2: A wallet owner wants to check their wallet's bitcoin balance

The second use case assumes that the user already has a working version of Bitcoin Core locally installed on their machine which is connected to a wallet. This use case is meant to outline the process of a user checking their wallet's Bitcoin balance through the Bitcoin Core GUI

application. First, the user presses a “verify balance” button in the App component’s GUI to ensure that they still have funds in their wallet. The App component will then connect to the RPC component, which subsequently establishes a connection to the peer network through the Connection Manager. Once a connection is established, the RPC component communicates to the wallet, asking it for its balance. The Wallet must first ask the storage engine to locate the balance on the local machine. Then, it returns the local balance to the RPC component through the Wallet component. Before reporting the wallet’s balance to the user, the system will verify that the local balance is equal to the expected wallet balance reflected by its transaction history on the blockchain. Doing this is required because a corrupted wallet must not be used to transact on the network to avoid bitcoin printing, erasing, and duplication. The process starts with the RPC component asking the Connection Manager component to verify the balance by connecting to available peers. The Connection Manager component does this by communicating with the Peer Discovery component which returns a connection to available peers. From here, the Connection Manager is now able to communicate with the Validation Engine Component. The Validation Engine component fulfills the request to verify the Wallet’s balance by checking to see if it matches with a calculated total from transactions associated with the wallet which are located on the blockchain. If the Validation Engine determines this is a valid Wallet balance, it is communicated back to the RPC, and subsequently visually displayed to the user. If the balances don’t match, the RPC component is told that the wallet is corrupted, and it updates the Wallet Component’s balance to the one reflected by the blockchain. This change is then propagated back to the user through the App component.

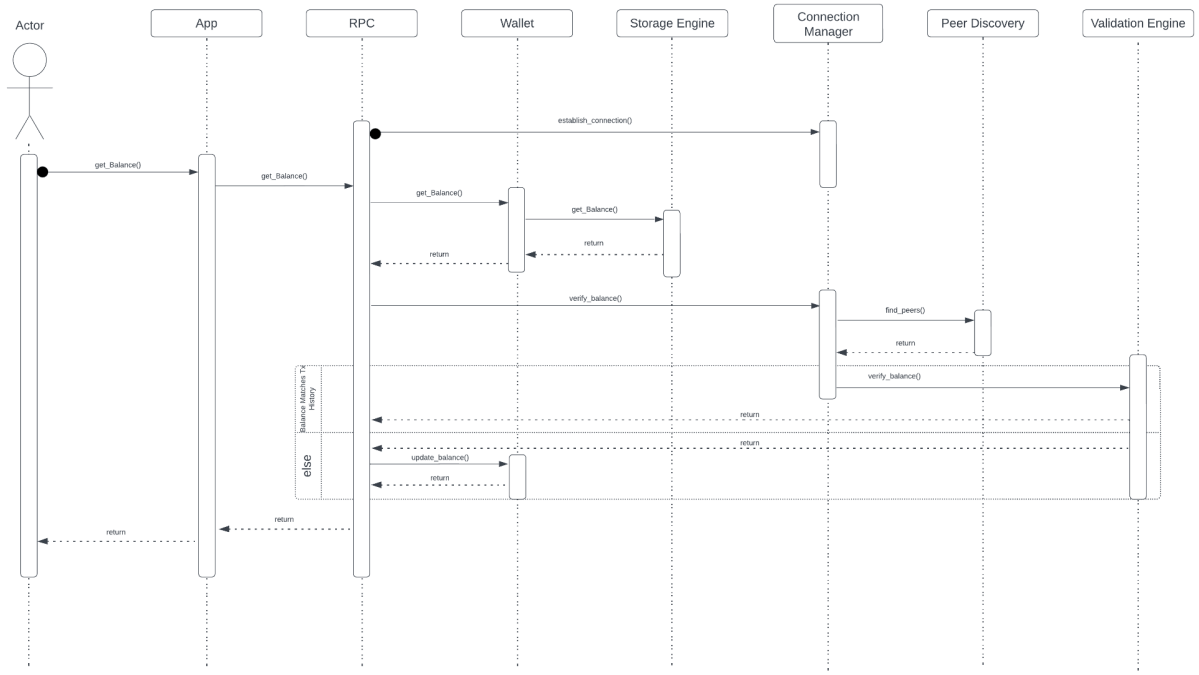

Figure 3: Sequence Diagram of Accessing Funds of a Wallet

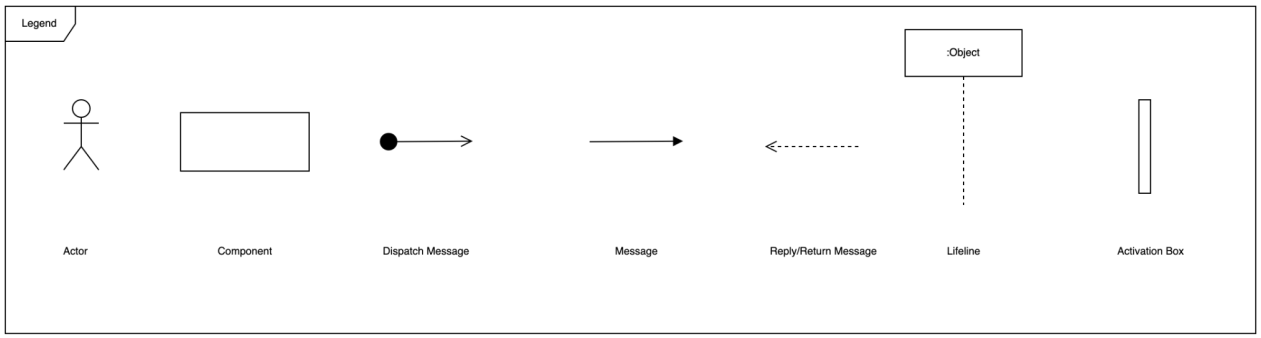

Figure 4: Legend for Sequence Diagrams

Evolution

Bitcoin Core is constantly being updated by the community of users through GitHub version control. The first step in implementing enhancements to Bitcoin Core is through filing of a Bitcoin Improvement Proposal (BIP). This document type describes proposed new features or changes to bitcoin's processes to the entire user base. There are three types of BIPs: standard, informational, and process. First, proposed changes in standard BIPs affect the network protocol, method of transaction, or block validation processes. Second, informational BIPs outline design issues, provide general guidelines, and supporting information, but do not propose new features. Lastly, process BIPs describe a Bitcoin process or propose a change to a process, such as changes to decision-making or development tools. Throughout the evolution of Bitcoin Core, the user base has revealed bugs and security vulnerabilities including buffer overflow and denial-of-service attacks. The evolution of Bitcoin Core discussed below excludes these routine bug fixes, new RPC commands, and mining difficulty increases. Alternatively, it provides a high-level overview of major changes such as feature introductions and improvements.

Bitcoin 0.1 - 0.8 (January 2009 - February 2013):

Released support for windows and added UNIX support with a cross-platform multi-core mining algorithm. Security safeguards that lock-in blockchain up to this point are included. Wallet private key encryption is introduced, allowing users to set a passphrase that must be entered before sending coins. Graphical user interface changes built on the Qt interface toolkit are now more intuitive.. Improved initial network synchronization times so new miners can integrate into the pool faster. Sending and receiving addresses now can be encoded into QR codes, allowing easier transactions. Bitcoin ownership can now be proved using a digital signature and wallets will be created with a smaller compressed public key, resulting in smaller transactions and less traffic on the network. A more efficient database system, LevelDB, is implemented to store transaction and block indices. CPU multithreaded transaction verification was also implemented and the transaction verification process was optimized allowing a running, synchronized node to use less memory. Finally, Mac and Windows binaries are signed with certificates, allowing them to be compatible with future OS updates.

Bitcoin Core 0.9 - 0.15 (March 2014 - September 2017):

Bitcoin Qt was rebranded to Bitcoin Core to reduce confusion between the Bitcoin network and client software Bitcoin Qt. This change included a new default transaction fee, more intuitive dependency manager, and an updated GUI which allows users to access new coin control features. New RPCs added to support a “headers-first” technique which optimized synchronization by making block downloads run in parallel. Transaction fees are estimated, allowing the user to choose whether they prefer a higher fee, faster confirmation time or lower fee, lower confirmation time. Wallets could now track transactions between known public addresses. Memory usage errors caused by low fee transaction flooding were combated, and anonymity through Tor network integration was improved. Also, support for Big Endian CPU architectures was implemented. Increased the speed of signature validation and security of nodes with low RAM allocation by using libsecp256k1 instead of OpenSSL and placing a maximum size on the mempool respectively. Nodes have lower storage requirements since they can prune locally stored historical block data. The transaction capacity of the network was increased using Segregated Witness which separates signature and transaction data, allowing more efficient use of block space. Hierarchical deterministic wallets are now supported to increase privacy and security, and the validation speed and network propagation performance is improved to achieve faster sync and block download times.

Bitcoin Core 0.15 - 0.20 (September 2017 - June 2020):

Users can now swap an unconfirmed transaction with a new, higher fee one in a redesigned section of GUI, and the ability to generate more private key pairs in a wallet increases security. Support for loading multiple separate wallets arrives, and coin age priority is removed, rendering the end of free transactions. Now, support for SigWit transactions is improved, increasing both the efficiency and security of the network. Further improvements to the fee estimation algorithm again decreased transaction fees. Support for Partially Signed Bitcoin Transactions (PSBT) was added which provides a more flexible way of allowing cooperation between signees. Wallet loading, creation, and unloading can now be done dynamically at runtime, simplifying the user experience. The program now defaults to native SegWit addresses in the Bech32 format, and more advanced and detailed documentation referring to JSON-RPC, REST, and PSBT interfaces and configuration files added to improve clarity for developers and users. OpenSSL completely phased out of use as a result of historical integration bugs, and the ability to import descriptors allows easier management of multi-sig and other wallet setups.

Bitcoin Core 0.21 - 24.0.1 (January 2021 - November 2022):

Introduction of descriptor wallets which store scriptPubKey information using output descriptors adds support for the Bitcoin scripting system. With this new wallet type comes a new database backend built on SQLite. Security and privacy increased by running Bitcoin Core as an I2P service and ensuring Tor v3 protocol is forced while on the Tor network. Also, connection process to peers optimized through addition of NAT-PMP port mapping support. Descriptor wallets

introduced in the last update are now default to improve the backup and recovery process for lost bitcoin funds. RBF transactions are now taken into account when calculating transaction fees, leading to a more accurate estimate. Additionally, taproot addresses are now natively supported for increased transaction security and scalability. Miniscript support was added to provide a toolbox for developers who are writing increasingly complex Bitcoin Script programs. Additionally, changeless transactions are introduced through a new RPC "sendall" which improves privacy and allows users to easily empty their wallet. Furthermore, change output values are now randomized by the wallet before informing its UTXO selection for spending which improves privacy by avoiding change fingerprinting.

Concurrency

Bitcoin Core demonstrates high concurrency through continuous and uninterrupted communication. This is essential for online transactions since maintaining reliability, stability, integrity, and security while being efficient, and convenient is demanded by the customer. Bitcoin Core runs on a variety of operating systems including Windows, macOS, and Linux, and implements concurrency in C++ by dividing its functions into separate subsystems that can execute concurrently. Each subsystem has a well-defined interface with other subsystems that ensures they can interact without conflict. Bitcoin Core employs a combination of POSIX threads library and a thread pool in a multithreaded architecture. Also known as "pthreads," they are used as a standard interface for creating and managing threads, and executing specific tasks in parallel such as communicating on the network. POSIX threads also provide a mechanism for synchronizing access to shared resources, such as the blockchain database, to prevent race conditions and ensure the integrity of the data. To avoid conflicts that could potentially lead to data corruption, Bitcoin Core uses locking techniques/mechanisms to coordinate the access of multiple threads to shared resources. On the other hand, the management of numerous tasks, such as block and transaction validation and their distribution across several threads is the responsibility of thread pools. This technique allows Bitcoin Core to eliminate the overhead of thread instantiation and destruction, enhancing its efficiency, resource management, security and stability.

Concurrency in the system also benefits the Storage, Mempool, Miner, and Validation Engine components. First, the Storage Engine utilizes multi threads to manage the blockchain database, allowing concurrent access to the database, while maintaining consistency and avoiding conflicts. The database is split into several files, each of which can be accessed independently, reducing contention between threads. In addition, locking mechanisms are used to coordinate access to the database and ensure data integrity. Second, the validation engine employs multiple threads to validate transactions and blocks concurrently. Each thread is responsible for validating a single transaction or block, and the results are combined once all threads have completed their tasks. This allows the system to process multiple transactions and blocks concurrently, improving overall performance and transaction throughput of the network. Third, the mempool is designed to operate on a separate thread in order to manage unconfirmed transactions independently. This allows the system to process multiple transactions and blocks concurrently, additionally improving the transaction throughput. Lastly, the Mining component uses multiple threads to mine new

blocks concurrently. Each thread is responsible for attempting to find a valid block hash by changing the block nonce value and the results are combined once all threads have completed their work. This allows the system to maximize the use of available resources and ultimately improve the speed and efficiency of the mining process.

Division of Responsibilities

The conceptual breakdown of the Bitcoin Core architecture infers that developers contributing to the project must have a clear division of responsibilities. Developers are organized into a team of core developers under the direction of senior core developers who are in charge of maintaining and optimizing the program. Each developer's role is assigned to certain areas of responsibility that pertain to their specialties. The core developers in charge of maintaining Bitcoin Core code are experienced members of the project. Their responsibilities include making major decisions about the software architecture, what features to include, how to handle security and scalability amidst recent rising popularity. Additionally, they are responsible for alignment with the consensus protocol. Most notably, core developers Wladimir Van Der Laan and Cory Fields have supported the production and successful deployment of new versions of Bitcoin Core. Other programmers on the team concentrate on particular components of the software such as the UI, mining software, and network protocol. These programmers may be in charge of adding new features, detecting and correcting bugs, and optimizing performance in their specialized fields. Alongside the core development team, there is a sizable community of developers that works on the project in a variety of ways. These developers might concentrate on individual tasks such as developing new wallets or programmes that communicate with the Bitcoin network. As an open-sourced project, users may contribute code to the main Bitcoin Core repository or test the software and give feedback to help find flaws and suggest optimality. Users outside the scope of the development team who wish to use Bitcoin Core are responsible for ensuring that they are utilizing the software effectively and safely. Correct use requires understanding the basics of how Bitcoin works (concepts such as private keys, public addresses, blockchains, etc.) and awareness of the security concerns related to its use such as the potential loss and access to funds if they are not adequately backed up and secured. Users should also maintain their software, keeping it up to date in order to prevent significant security and bug vulnerabilities. They should think about their own security and privacy (i.e. using the software in conjunction with a trusted hardware wallet, or taking other precautions to ensure transactions retain privacy and security). Ultimately, users can contribute to the safe, dependable, and effective operation of Bitcoin Core by taking these obligations seriously.

External Interface

The external interface of the Bitcoin Core provides the pathway for users to interact with the system and access Bitcoin network information. The Graphical User Interfaces (GUIs) allow users to view and manage their Bitcoin wallets and send and receive Bitcoin transactions. The GUI displays information such as the user’s transaction history and current balance. Bitcoin Core uses

files to store the Bitcoin blockchain information, the network, and the client's configuration settings.

Lessons Learned

Throughout the research process, we found the documentation very informative, clearly outlining the architecture style used and the central components that form the system. Without viewing the source code of Bitcoin Core, we found it became difficult to determine the individual purposes for each component. Since there is a plethora of documentation on how Bitcoin systems are composed from the Bitcoin Developer resource, we found that the interaction between the components was fairly standard in Bitcoin Core. However, since some group members had little prior knowledge of blockchain technology, complementary research was required to understand the transaction process and how it is completed. Despite this initial hurdle to be overcome, we saw success in our weekly meetings and used social media to fluidly communicate with each other. Additionally, we utilized a shared document to synchronize research completed by team members. By communicating effectively and working collaboratively, we found that we were able to swifty make decisions to meet self-imposed deadlines as questions and discussion arose.

Conclusion

In conclusion, our conceptual architecture of the Bitcoin Core demonstrates the inner workings of how the software's essential components interact in order to work as a decentralized digital currency system that uses a peer-to-peer architecture to facilitate secure and direct transfers of value. Our report looks at the many system components and how they work together, as well as how the software has changed over time and how it has affected outside interfaces and developers. The report offers use cases to back up the structure.

Data Dictionary and Naming Convention

Term	Definition
Node	Computer connected to other computers, following rules and sharing information
Unspent Transaction Output	Amount of digital currency authorized by one account to be spent by another
Blocks	Set of Bitcoin Transactions
Blockchain	Collection of Blocks stacked on top of eachother

Acronym	Full Context
UI	User Interface
CLI	Command Line Interface
API	Application Programming Interface
UTXO	Unspent Transaction Output
OS	Operating System
P2P	Peer-to-Peer
RPC	Remote Procedure Call
RBF	Replace-By-Fee
QR Code	Quick Response Code

References

Bitcoin Core. (2019). Version History. Bitcoin.org. Retrieved February 15, 2023, from https://bitcoin.org/en/version-history

Nakamoto, S. (2002). *Open source P2P money*. Bitcoin. Retrieved February 15, 2023, from https://bitcoin.org/bitcoin%20pdf

Unknown, U. (2009). *Bitcoin Core Validation*. Validation - Bitcoin Core Features. Retrieved February 15, 2023, from https://bitcoin.org/en/bitcoin-core/features/validation

Team, B. (2011, September 15). *Lan 101: Networking basics*. Tom's Hardware. R. 02/15/2023, https://www.tomshardware.com/reviews/local-area-network-wi-fi-wireless,3020-2.html

Unknown. (2017). *Object Oriented Architecture* . Object oriented architecture. 02/15/2023, from https://www.tutorialride.com/software-architecture-and-design/object-oriented-architecture.htm

Saini, D. (2013, December). *Security concerns of object oriented software architectures - researchgate*. Security Concerns of Object Oriented Software Architectures. Retrieved February 15, 2023, from https://www.researchgate.net/publication/259470705_Security_Concerns_of_Object_Oriented_Software_Architectures

BitStamp. (2022, August 17). *What is a bitcoin improvement proposal? (BIP) - bitstamp learn center*. What is a Bitcoin Improvement Proposal? (BIP). Retrieved February 16, 2023, from https://www.bitstamp.net/learn/blockchain/what-is-a-bitcoin-improvement-proposal-bip/