Bitcoin Core's Infrastructure
A Concrete Architecture Report

CISC 322
Software Architectures
Professor Bram Adams

Group 34:
Sweet Home Ala-bram-a

Jean-Philippe Le Blanc - 20157203 - 18jpcl@queensu.ca (Team Leader)
Dallas Doherty - 20220683 - 19ded2@queensu.ca (Presenter)
Ethan Mah - 20224551 - 19ettm@queensu.ca (Presenter)
Ruairí O'Connor Clarke - 20235043 - 20romoc@queensu.ca
Muhammad Ibrahim - 20218324 - 19mi22@queensu.ca
Mayy Mounib - 20230803 - 19mlm15@queensu.ca

Submitted on March 24th, 2023

Table of Contents

Abstract

This technical report will provide a detailed overview of the concrete architecture of Bitcoin Core. After investigating the source code using Understand and our own Bitcoin Core research, subsystems that we did not identify in the conceptual architecture, such as Nodes, were added. Additionally, five previously predicted subsystems from our conceptual architecture were removed such as Peer Discovery, Local Peers, Local Storage, and Blocks. Predicted dependencies in the conceptual architecture were removed and additional dependencies were discovered in the concrete. Using the source code, these additional dependencies were mapped to our concrete architecture. Our reflexion analysis of both the high-level architecture and the inner architecture of the wallet subsystem helped us gain insight into the areas where our predictions were incorrect. We found out that the initial proposed conceptual architecture was incomplete since it did not provide several of the dependencies that were essential in the concrete architecture. However, our conceptual architecture did predict most of the subsystems needed in our concrete architecture.

Introduction

In 2008, Satoshi Nakamoto sought to create a decentralized, digital currency that would allow for secure, transparent, and fast peer to peer transactions without a central authority. Bitcoin, the first and most well-known cryptocurrency, has revolutionized the way we think about currency and transactions. At the heart of the Bitcoin system is Bitcoin Core, an open-source software that runs as a full node and provides the backbone for the decentralized P2P network that powers the Bitcoin network. The architecture of Bitcoin Core is crucial to understanding how Bitcoin operates. This report investigates the concrete architecture of Bitcoin Core, examining its subsystems and their interactions. It contains an explanation into how we derived our final concrete architecture and revised our conceptual architecture. The report has five main sections. The first section dives into our concrete architecture explaining its subsystems and dependencies within subsystems (see Figure 1).

The second section documents the derivation of our concrete architecture at a high level as well as uses a reflexion analysis to explain the differences between our concrete architecture and our conceptual architecture. Using documentation and our own Bitcoin Core research we derived our original conceptual architecture (see Figure 2). This figure describes most of the high-level subsystems and dependencies seen in our concrete architecture. The analysis outlines the discovery of unexpected dependencies, removal of subsystems, and addition of new subsystems. These dependencies are explained using files, components, and source code.

The third section documents the subsystem decomposition of the Bitcoin Core wallet. This section explains the wallets subsystems and dependencies in detail. This section investigates the inner architecture of the module using a more in depth reflexion analysis on this particular subsystem (see Figure 3 and 4). The analysis takes Group 12's conceptual architecture of the wallet and compares it with our concrete architecture.

The fourth section presents two sequence diagrams (see Figure 5 and 6). for non-trivial use cases. This section aims to show how each part interacts with others. The first scenario is a use case diagram for transactions. Outlining the components of the transaction and how they interact with each other through a timeline. The second scenario is a use case diagram for adding a block to the blockchain. It also outlines the components of the action and how the subsystems interact to perform it.

The final section summarizes the report with a conclusion and outlines the lessons learned from the report. This section reviews limitations in our analysis and thoughts to improve the architecture for the future. This section is then followed by our references.

Concrete Architecture

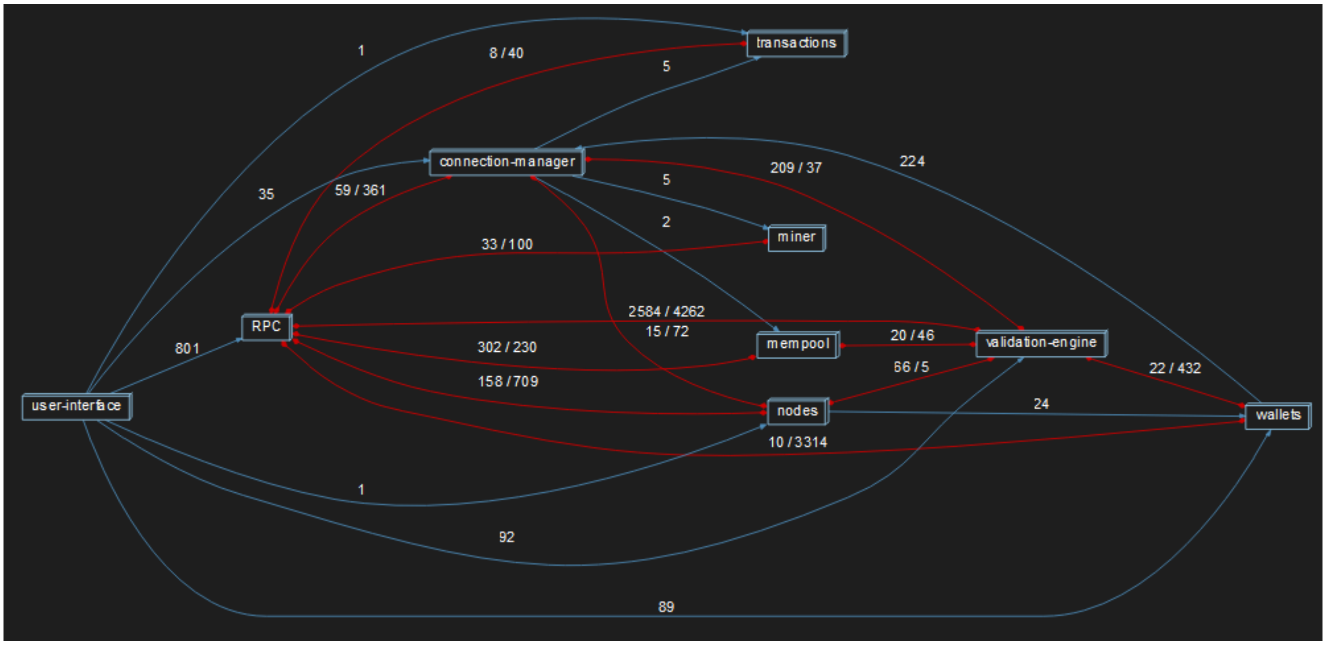


Figure 1: Bitcoin Core's concrete architecture from Understand.

Bitcoin Core's concrete architecture implements a peer-to-peer (P2P) architectural style. This aligns with the conceptual architecture as this is the only style of architecture that allows the system to achieve its goal of having a completely decentralized monetary system. The Nodes within the P2P system are the miners and the nodes, who are both tasked with validating and maintaining the system through a series of complex algorithms. It also implements aspects of a Publish/Subscribe (pubsub) system in order to propagate events throughout the system. The Remote Procedure Call (RPC) subsystem is what takes charge of the pubsub aspect of the system. We can see that in the Understand diagram, the RPC connects many subsystems. It can be seen as one of the means of passing messages from one system to another when a meaningful event occurs.

Within the concrete architecture there are many subsystems that must interplay in order for the complete system to work smoothly. Each subsystem takes care of their respective aspect of the system, but they must communicate with other subsystems. We can see here what the interactions are with the other parties:

User-Interface: The graphical user interface (GUI) or command-line interface (CLI), which comes as part of the Bitcoin Core software package, offers a visual representation of the node and wallet functionality Bitcoin Core offers. This is the starting point for users to interact with the system and perform transactions with other peers. The user-interface interacts with the following subsystems.

- RPC: This connection to the RPC is to propagate messages to the entire system when a transaction occurs.
- Wallets: Allows for the UI to retrieve their current balance and more.
- Nodes: Connects to the current instance node.
- Connection-Manager: Allows for peer discovery of neighboring nodes.
- Validation-Engine: Allows for the user to connect to their wallet.
- Transactions: Allows for the UI to retrieve the users previous transactions.

Nodes: A discrete member of the network which interacts with other nodes to form the network. Bitcoin nodes store and validate the blockchain and exchange blocks and transactions with one another to maintain consensus. Within each node there is the Local Storage, that stores ledger, node users wallet information. The nodes interactions with other subsystems are as follows:

- Connection-Manager: The connection manager allows for nodes to discover other peers on the network and to communicate with them.
- Validation Engine: Used to validate the node in the system and to validate anything coming out of the node.
- Wallets: The nodes have access to their respective wallets.
- RPC: Allows the node to propagate messages throughout the system.

Remote Procedure Call (RPC): Allows other programs to control Bitcoin Core, including the ability to spend funds from your wallets, affect consensus verification, read privacy data, and otherwise perform operations that can cause loss of money, data, or privacy. This can be understood as the publish/subscribe aspect of Bitcoin Core, as it is in charge of propagating messages and data throughout the system. The RPC utilizes the following subsystems:

- Mempool: Updates the mempool with the new events that it received.
- Miner: Allows the miner to notify the system that a block has been validated.
- Validation-Engine: Validates the message and uses the algorithms for secure transmission
- Nodes: Allows the nodes to communicate with the neighboring nodes.
- Transactions: Notifies the network of new transactions that occurred.
- Connection-Manager: Notifies the connection manager upon new events.

Validation-Engine: Contains cryptographic algorithms. The algorithms contain scripts for execution and caching, data obfuscation and hashing. The script subtree is defined here which contains procedures for defining and executing Bitcoin scripts as well as signing transactions. Defines how we update our validated state. Handles modifying data structures for chainstate and transactions (mempool) on the basis of certain acceptance rules. The validation engine interacts with the following subsystems:

- Mempool: Validates that the mempool is correct and up to date.
- Nodes: Validate that a node is not harming the system and the work it is doing.
- Wallets: Validates that the content of the wallet is up to date.
- Connection-Manager: Validates the connections between two nodes.
- RPC: Allows to propagate to the network when an event is valid/invalid.

Wallets: In charge of generating and storing public and private keys. In addition it is where the users Bitcoins are stored. The user can then start, receive and accept/reject transactions. The wallets use the following subsystems:

- RPC: Propagates the updated wallet when an event occurs within the wallet.
- Connection-Manager: Connects neighboring nodes to the wallet.
- Validation-Engine: Validates the current balance and wallet is valid on the blockchain.

Connection-Manager: Handles network communication with the P2P network. It contains addresses and statistics for peers that the running node is aware of. It manages socket communication and network interactions more generally for each peer. The connection manager has the following interactions:

- Nodes: The connection manager allows for nodes to discover the neighboring nodes in the network.
- Mempool: The connection manager only communicates with the mempool to register it in the possible connections.
- Validation-Engine: This subsystem is used to validate whether a connection is valid or not.
- Transactions: The connection manager keeps track of the transactions and works with the RPC to send the information.
- RPC: The RPC is utilized by the connection manager to propagate information to the network.
- Miner: The miners are used by the connection manager to know what miners are doing work.

Transactions: Provides a definition for the in-memory data structure that manages the set of transactions this node has seen. This data structure provides a view of transactions sorted in different ways (i.e. by fee rate, entry time, etc.). The mempool is of fixed size, so eviction logic is defined here too. As well, as an index of the UTXO set which includes unconfirmed mempool transactions. The transaction subsystems interacts with the following subsystems:

- RPC: Transactions are broadcasted using the RPC to the network to notify other peers to start validating the transaction.

Miner: This subsystem includes utilities for generating blocks to be mined and for adding them to the blockchain. This is achieved in Bitcoin Core by continuously attempting to solve a complex mathematical problem called the proof-of-work algorithm. Once a new block has been created, the miner notifies the rest of the network using the RPC subsystem. The miner subsystem also takes care of any task related to mining, such as monitoring integrity of the blockchain and managing hardware. The miners interact with the following subsystems:

- RPC: The miners broadcast to the network when a new block is created using the Remote Procedure Call subsystem.

Mempool: The waiting area for bitcoin transactions that each full node maintains for itself. After a transaction is verified by a node, it waits inside the mempool until it is picked up by Miner and inserted into a block. This can be understood as the staging area before a change is made to the main transaction ledger. The mempool has interactions with these subsystems:

- RPC: Allows to communicate with other nodes the updated and validated blockchain.
- Validation-Engine: Gives access to the algorithms needed to validate transactions added to the mempool.

Derivation Process of High Level Architecture

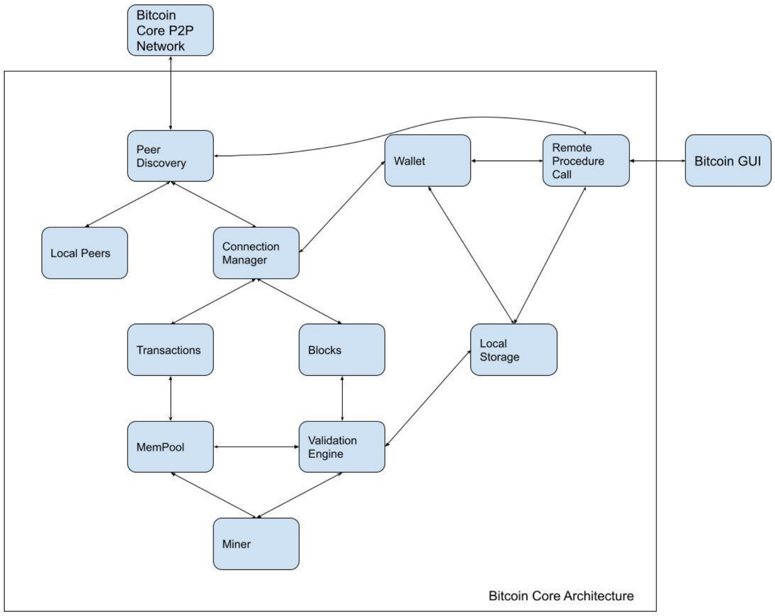


Figure 2: Bitcoin Core's Conceptual Architecture from the Conceptual Architecture Report.

We began our derivation process by reviewing our original conceptual architecture of the Bitcoin Core System from our previous conceptual architecture report. After close examination, our Bitcoin Core Conceptual architecture was a great starting point, however it did not represent the actual concrete architecture. As a group we gathered together and analyzed what components of the Bitcoin Core system relied and depended on one another through a series of code reviews and our own research. With the help of the Bitcoin Core documentation and peer to peer network visualizations we were able to derive our new concrete architecture using Understand. Once we imported the source code into the Understand software, it enabled us to see the relationships in the Bitcoin Core system without skimming through the source code in fine detail. Subsequently, we were able to supplement our concrete architecture and fix any dependencies that did not make sense. If we were ever unsure of the subsystem a file belonged to we made sure to use a helper subsystem to locate the dependencies of the file and decide on the subsystem that file most likely belonged to. Compared to our conceptual architecture new dependencies were added, and some of our conceptual dependencies were removed. These changes will be discussed in further detail in our reflexion analysis. Ultimately, our generated concrete architecture confirms our previous inference of the Bitcoin Core architecture style that is P2P and has left us all with a greater understanding of the internal relationships of the Bitcoin Core system.

Reflexion Analysis of High Level Architecture

In our conceptual architecture we had the miner subsystem having a dependency on the Mempool so that the miner could access the unconfirmed transactions when creating a block. In the concrete architecture we found that this dependency was not there as the miner instead depended on the connection manager for information pertaining to transactions to be added to blocks. This also explains the new dependency found between the miner system and the connection manager as the connection manager needs to communicate with the miner to get information related to blocks and transactions being added to a block. In the conceptual architecture, the validation engine and the miner both depended on each other; however, in the concrete architecture this dependency was not found. This dependency was put in conceptually for the miner to be able to validate transactions going into a block; however in the concrete architecture we found this was not needed as the connection manager is instead calling on the validation engine when there are transactions to validate.

The transactions subsystem had a new dependency on and from the RPC subsystem in the concrete architecture. This unexpected dependency is because the RPC makes calls to get the hash of a block that a transaction is being added to. In the conceptual architecture we had a peer discovery subsystem as well as a connection manager, we found that in the concrete architecture the peer discovery system was redundant as the connection manager handled its functions.

In the conceptual architecture the wallet subsystem only depended on the connection manager and was depended on by the local storage (changed to be called nodes in concrete architecture). Concretely, the wallet system also depended on RPC, the validation engine and the RPC. This is because the wallet contained functions pertaining to peers, consensus validation and accessing the UTXO set. In our conceptual architecture we had a subsystem for blocks; however, when we got to the concrete architecture the blocks subsystem was absorbed by other systems which carried out its functions separately.

Detailed Explanation of The Bitcoin Core Wallet

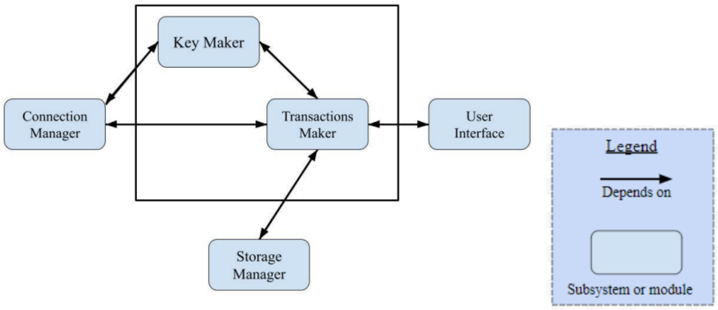


Figure 3: Bitcoin Core's Conceptual Architecture for the Wallet Feature from Group 12.

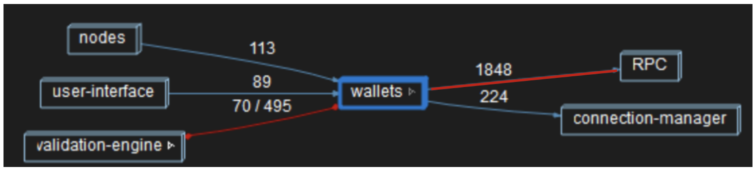


Figure 4: Bitcoin Core's Concrete Architecture for the Wallet Feature.

As previously discussed in assignment one's report, the Bitcoin Core wallet manager is responsible for managing the user's cryptocurrency balance and facilitating transactions by interacting with the peer-to-peer network. To accomplish this, it receives the latest updates from the blockchain and broadcasts new transactions accordingly. With this understanding, we have conducted a more in-depth investigation of the wallet subsystem using the Understand software. Following similar steps from our derivation process, we performed a code analysis to identify the relationships between the interactions in the wallet subsystem.

As can be seen in Figure 4, the connection manager demonstrates a one-way dependency on the wallet, while the wallet depends on the nodes and the UI interface. Additionally, a mutual dependency is present between both the wallet and the validation engine and RPC. More specifically within the wallet, there is a function module which houses code files like wallet_balance and wallet_loading that are responsible for management tasks such as loading the wallet into the memory, checking the balance, and creating and funding transactions. This module is coupled with the main Bitcoin wallet program which, in turn, handles command-line arguments and the user interface to effectively manage the wallet. On the other hand, as we examine its conceptual architecture adopted from Group 12, we can identify certain discrepancies between the features present in the conceptual architecture and those in the concrete architecture. These discrepancies will be discussed further in the reflexion analysis below.

Furthermore, although the entire Bitcoin Core system demonstrates a peer-to-peer architecture style, the Bitcoin wallet subsystem interacts with its various components through layers. The lower-level layer, which includes the connection manager, RPC, and crypto-library, is responsible for communicating with the peer-to-peer network. Similarly, the wallet layer consists of the function modules discussed above which ensure the proper functioning of the wallet. Together, these components make up the detailed architecture of the wallet subsystem in Bitcoin Core, ensuring that users can securely store and transact with their cryptocurrency.

Reflexion Analysis of Subsystem – Bitcoin Wallet

In reference to group 12's conceptual architecture model, the wallet subsystem has dependencies on three external components that connect a given node's information to Bitcoin Core's network and the wallet subsystem to the user interface. The system begins with a given node's private key, using the connection manager component it connects to the Bitcoin network to request their wallet information (transactions, Bitcoin currency, etc.). Upon connection, a key and transaction maker connect to Bitcoin's internal storage to retrieve a user's information using the storage manager component. Once the information has been retrieved from the server, the wallet subsystem connects to the user interface to display the results and information to the user.

In Bitcoin Core's concrete architecture model, there are additional components and dependencies from the wallet subsystem. Similar to the conceptual architecture, there remains a connection-manager, user-interface, and crypto-library (storage manager); however, additional features were overlooked in the Bitcoin wallet conceptual architecture.

In the concrete model, the wallet has a security protocol before being connected to the Bitcoin Network, using the validation-engine to verify access to a node's information. The requirements are heavily dependent on the form of security but can range anywhere from a unique password to access keys and two factor authentication. Similar to the conceptual concept, the connection-manager connects the wallet subsystem to the Bitcoin network and accesses the crypto-library and local storage to access the necessary information for the wallet. This was another discrepancy as the need for local storage information access was overlooked during initial design and the proper naming convention used for the "storage manager".

Lastly the connection between a node's local system, the wallet subsystem, and Bitcoin Core's software was understated in the conceptual architecture. As displayed in the concrete model, the wallet has a great number of dependencies on the API and RPC before displaying information through the user interface. These dependencies can be explained as the set of protocols and routines run by the program to other networks and subsystems that are needed to retrieve wallet information, security validation, and application tools.

Sequence Diagrams

Use Case 1 - Transactions:

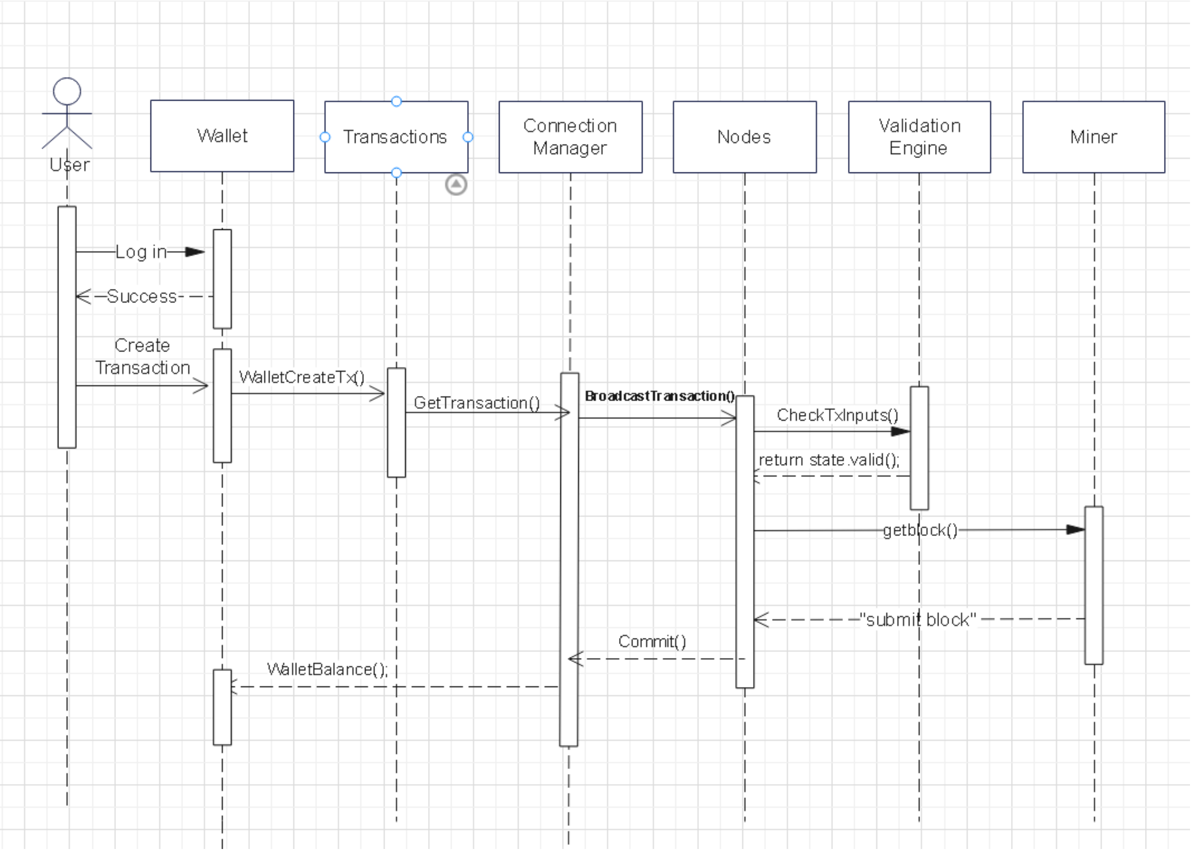


Figure 5: Use-Case diagram for transactions.

Once a user starts up their local bitcoin core wallet they can create a transaction, specifying the amount to send, receiver address and the fees they're willing to pay. The wallet system calls on WalletCreateTx() which calls on the transactions subsystem to create a new transaction, transactions calls GetTransaction() to communicate the new transaction to the connection manager which in turn, calls BroadcastTransaction() to send the transaction to the nodes in the network, each node then validates the inputs and formatting of the transaction by calling CheckTxInputs() to access the validation engine which returns a boolean validity. The node then uses getblock() to get a block from a miner for the transaction to be added to. The miner then informs the nodes that the transaction has been added to a block. The nodes use Commit() to confirm the transaction and the block by consensus with the other nodes. Finally, the connection manager calls on WalletBalance() to update the user's wallet to reflect the transaction.

Use Case 2 - Adding a block to the blockchain:

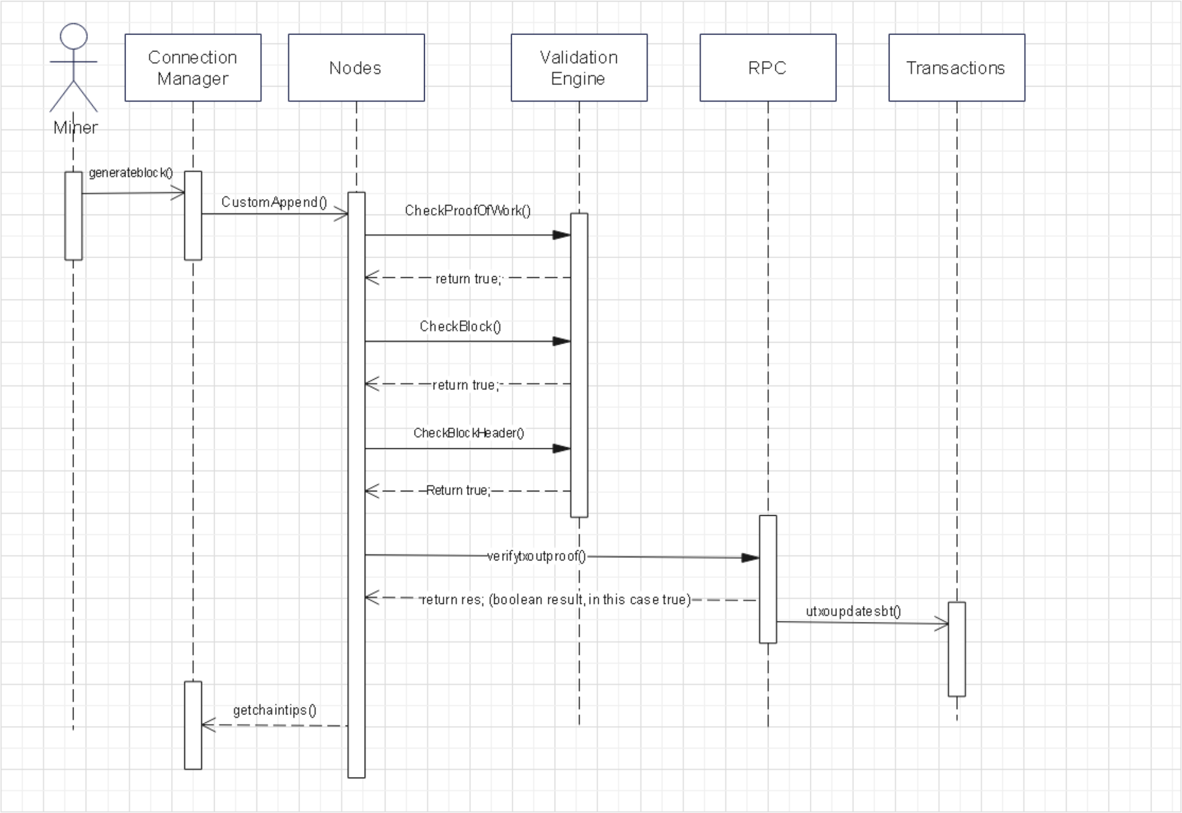


Figure 6: Use-Case diagram for adding a block to the blockchain.

Any block being added to the blockchain must first be validated by every node in the peer to peer network. Once a miner has created a new block, generateblock() will be called to broadcast this block to the network. After being broadcast, the connection manager will use CustomAppend() to add this new block to the nodes in order for it to be validated by each. The nodes will communicate with the validation engine to perform a series of checks on the new block. The first check made by the validation engine is CheckProofOfWork() which tests that the block has a valid proof of work with the necessary difficulty. Next, the block's formatting must be checked, this includes inputs, outputs and fees in the transactions within the block, to do this the nodes call CheckBlock() to access the validation engine and validate its formatting. The header of the block must then be verified against the header of the previous block to ensure it's being added properly, to do this CheckBlockHeader() is called. After all these have been validated the nodes must use the Remote Procedure Calls to verify that the outputs for the transactions in the block are real and unspent, to do this, nodes calls verifytxoutproof(). Once the transactions' outputs have been verified RPC will make a call to transactions using utxoupdatepsbt() in order to remove the outputs from the UTXO set as the transactions have now been at least partially signed. Finally, the node calls getchaintips() to check the current end of the blockchain and add this new block to the tip and broadcast it to the network through the connection manager.

Conclusion

This technical report delivers the concrete architecture of Bitcoin Core. The system implements a peer-to-peer (P2P) architectural style, which allows for a completely decentralized monetary system. Additionally, a Pub-Sub system is used to propagate events throughout the system, with the Remote Procedure Call (RPC) subsystem handling the pub-sub aspect. The report outlines the interactions between the different subsystems, including the Node, Validation Engine, Mempool, Wallet, Connection Manager, and Miner. The report highlights the importance of nodes in the network, as each individual instance of the Bitcoin Core software is responsible for both sending and receiving data, and controlling the system. The report also discusses the derivation process, where the original conceptual architecture was reviewed and analyzed to create a new concrete architecture. This process involved importing the source code into the Understand software to visualize the relationships between the different subsystems, and supplementing the architecture to fix any dependencies that did not make sense. We concluded that the P2P architectural style is best for the concrete and conceptual architecture of Bitcoin Core. To obtain the concrete architecture we reviewed our conceptual architecture in the last report. We noticed unnecessary dependencies and subsystems using a graphical representation of our P2P system, and developers documents. Our reflexion analysis discovered new dependencies between multiple subsystems. Through our trials and tribulations of translating our idea of what Bitcoin Core was to an actual Conceptual then Concrete software architecture we have come to know a deeper understanding of the inner workings of the Bitcoin system, and the idea of cryptocurrency as a financial system.

Lessons Learned

A primary lesson learned by completing this report was understanding the inherent complexity associated with a system of this scale. With numerous interdependent subsystems and intricate systems in place to ensure proper functioning, this complexity was not unexpected, but delving into the source code emphasized its magnitude. In Particular, we realized the significance of comprehensive documentation when managing large-scale systems. We also gained an appreciation for the importance of reverse engineering in the process of understanding how subsystems interact and depend on one another, particularly when grappling with inter-subsystem dependencies. Finally, another key takeaway was that the Understand software crashes frequently when big modifications are made, so it was best to make small incremental changes in order to not overload the program.

Data Dictionary

Peer-to-peer (P2P): a computer network in which each computer can act as a server for the others, allowing shared access to files and peripherals without the need for a central server.

Proof-of-work (PoW): A consensus mechanism where users must solve a cryptographic puzzle to get a reward.

Unspent Transaction Output (UTXO): is the technical term for the amount of digital currency that remains after a cryptocurrency transaction.

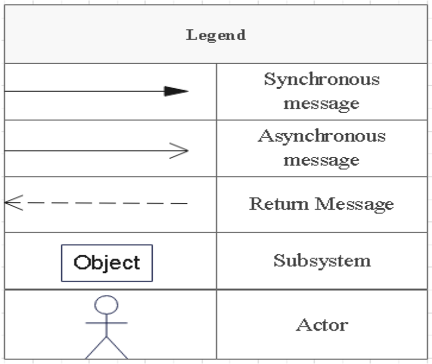
Graphical User Interface (GUI): A graphical way to interact with a system.

Command Line Interface (CLI): A textual way to interact with a system.

Application Programming Interface (API): It is a set of protocols, routines, and tools for building software applications that specify how software components should interact with each other.

Remote Procedure Call (RPC): A protocol used by computer programs to request a service from another program running on a remote system over a network.

Use-Case Diagram Legend:



Legend	
——►	Synchronous message
——>	Asynchronous message
<– – – – – – –	Return Message
Object	Subsystem
	Actor

References

"Dev++ 02-22-EN | An overview of Bitcoin Core architecture - James O'Beirne | Platforms." Bitcoin Edge, 3 November 2018,https://www.youtube.com/watch?v=L_sI_tXmy2U.

"Bitcoin Mempool." 99Bitcoins, n.d., https://99bitcoins.com/bitcoin/mempool/.

"Bitcoin-qt." River Financial, n.d., https://river.com/learn/terms/b/bitcoin-qt/.

"Directory Hierarchy." Doxygen Bitcoin Core Documentation, n.d., https://doxygen.bitcoincore.org/dir_68267d1309a1af8e8297ef4c3efbcdba.html.