# Conceptual Architecture on Bitcoin Core

Group 20 - GoDaddy

Junhui Zong
Brian Cho
Julian Wei
Yuchen Li
Ye Shen
Arsh Kochhar

## Abstract

The purpose of the study was to extract the conceptual architecture of a large software system. Bitcoin Core was selected for analysis which involved deriving the subsystems and its dependencies. Our studies found that Bitcoin Core is an open-source implementation of the Bitcoin protocol that mainly utilizes a P2P (peer-to-peer) architecture style. We also found that it uses a Layered architecture style that is made up of the Interfacing layer, Logic layer, and Data layer. We found that the program uses a Client-Server architecture style as well, leading to the discovery that the P2P and Client-Server style can co-exist. By analyzing the subsystems of Bitcoin Core, we gradually understood their respective characteristics and functions, as well as how they are related to other subsystems. It was concluded that the relationship between *Validation Engine* and the other subsystems make Bitcoin Core a secure software platform for managing Bitcoin transactions.


## Introduction and Overview

The emergence of Bitcoin as a decentralized digital currency has led to an increased interest in the underlying technology that makes it possible. Bitcoin Core is the implementation of the Bitcoin protocol. The program is responsible for the creation, storage, and management of Bitcoin transactions. To understand the design of the system, it's essential to examine its conceptual architecture, which describes the high-level design and organization of the system.

This report provides an in-depth analysis of the conceptual architecture of Bitcoin Core, including its various subsystems that deal with networking, consensus, and wallet management. The report also discusses Bitcoin Core's conceptual architecture diagram that was derived using the layered architecture style.The report also presents a couple use cases that highlight the relevance and practical applications of the system.

## Architecture Styles

Bitcoin Core is mainly structured using the P2P architecture style where nodes in the network both provide and consume services at the same time. A Bitcoin node is a program that implements the Bitcoin protocol.

Bitcoin Core also follows the Layered architecture style. The main layers of the Bitcoin Core architecture can be broadly classified as Interfacing layer, Logic layer, Database layer.

The Interfacing Layer has two subsystems: *Peer Discovery and RPC*. Additionally, it oddly includes the database *Peers*, which contains the peers that the node is aware of. Both *Peer Discovery* and *RPC* can interact with the subsystems in the layer underneath. The two subsystems are responsible for connecting with the Bitcoin network and user interactions with the system.

The Logic Layer has six subsystems: *Connection Manager, Mempool, Wallet, Miner, Validation Engine,* and *Storage Engine.* The *Connection Manager* and *Wallet* interact with the layer above. The *Storage Engine* interacts with the layer underneath. The subsystems in this layer are responsible for accomplishing functional requirements such as validating transactions, mining, etc.

The Database Layer has three databases: *Headers, Blocks, Coins*. Each database interacts with the layer above.

Bitcoin Core follows the Client-Server architecture style. The external applications operate as clients to request the data from Bitcoin Core, which acts as the server. The client interacts with the server through the RPC interface.

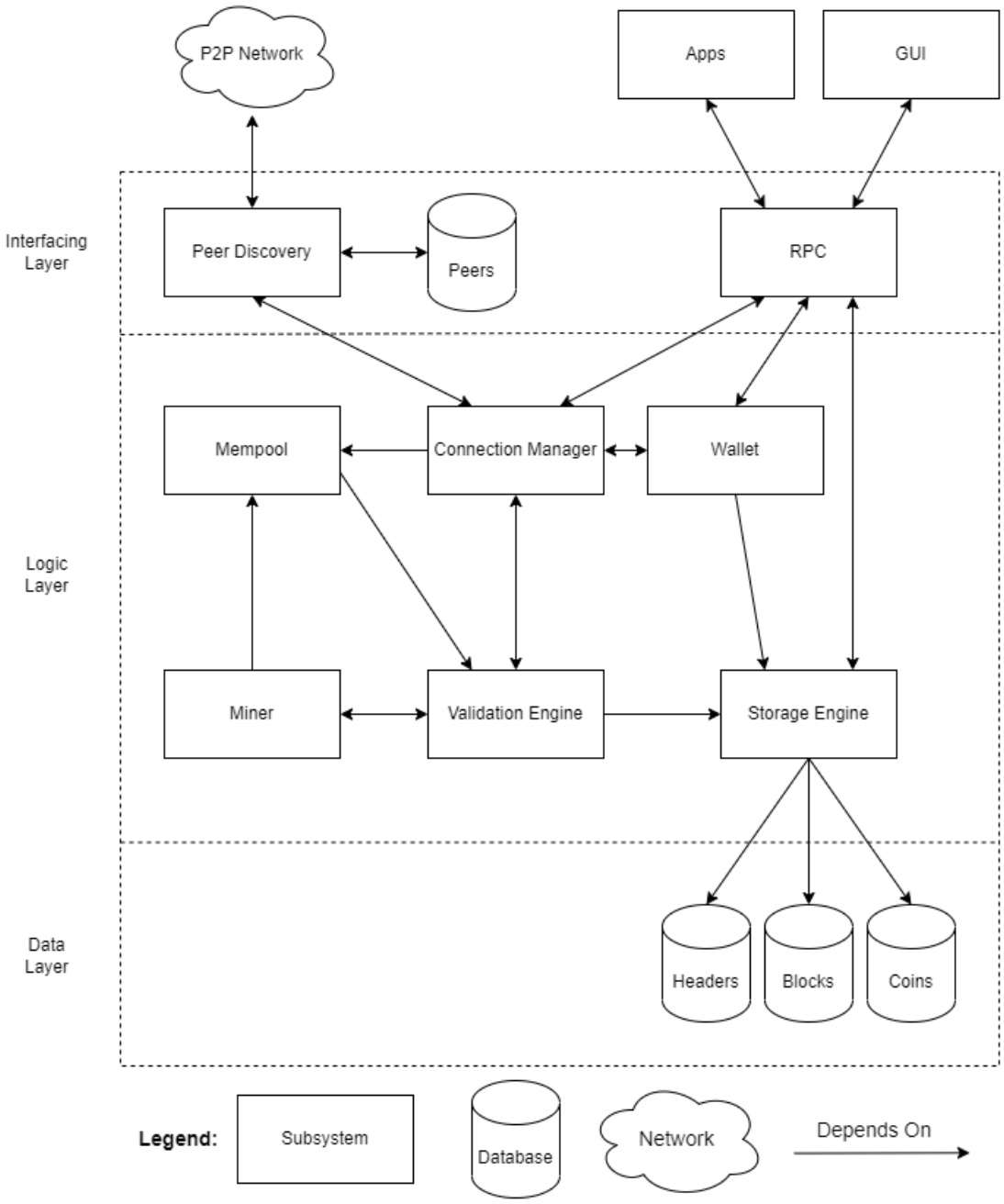


Figure 1: Conceptual Architecture Diagram

## Derivation Process

The derivation of the diagram started with the diagram provided in the open-source book provided in the project description [1]. This diagram provided us the foundational view of Bitcoin Core's architecture.

One improvement we made is removal of the Txs and Blocks components below the Connection Manager. The reasoning for this removal is that these two components were there to show that only transactions were going to the *Mempool* and only blocks were going to the Validation Engine. This removal made sense to keep unimportant components off the conceptual diagram. Another improvement we have made is the separation of the subsystems in their respective tiers. This helped to better illustrate the main functionalities of various subsystems, with clear separation of systems which dealt with external interfaces, internal logic, and storage.

## Alternatives

Though we heavily used the reference diagram, we still had some alternate designs. One design we had in mind was a repository style that was centered on a central database containing the entire blockchain. Components would act upon the database with each component accessing a specific part of the database. We decided to discard this design model since there are components present in Bitcoin's documentation that have no relations to any database. One such component is the miner. There are also implementation problems, if components only need to access parts of the central database why not have many separated databases.

# Architecture

This section discusses the various subsystems that our architecture relies on. They are *Peer Discovery, Connection Manager, Wallet, Validation Engine, Mempool, Storage Engine, Miner, and Remote Procedure Call (RPC).*

## Peer Discovery

Subsystem *Peer Discovery* serves as the point of contact between the Bitcoin network and the Bitcoin Core node. Its first responsibility is the acquisition and storage of IP addresses of other peer nodes in the network. Its second responsibility is relaying incoming data from the network to the *Connection Manager* and also broadcasting outbound data from the *Connection Manager* out onto the network.

*Peer Discovery* is heavily involved in discovering IP addresses of peer nodes in the Bitcoin network.

*Peer Discovery's* other responsibility is the receiving and broadcasting of data. Any incoming data, like initial block download requests from another node will be received by *Peer Discovery* and transmitted to the *Connection Manager*. Any outbound data, like a validated block mined by the node's *Miner* will be broadcasted by *Peer Discovery.*

The *Peers* Database stores the IP address of all known peer nodes. *Peer Discovery* stores any newly discovered peer into this database. *Peer Discovery* accesses this database whenever it needs the address of one or more known peers.

## Connection Manager

Subsystem *Connection Manager (CM)* serves as a central controller of the architecture that controls the flow of data. Data could be of many different forms, like messages sent by another peer or a new block sent by a miner. The way *CM* handles its responsibility is by reading each type of inbound/outbound data and reacting accordingly. That's why *CM* interacts with many different components. It interacts with *Peer Discovery, Wallet, RPC, Validation Engine, and the Mempool.*

## Wallet

The wallet in the Bitcoin Core architecture is a software component that is used to store, manage, and track the ownership of bitcoins. It is responsible for generating and storing a pair of cryptographic keys, which consist of a public key and a private key. The public

key is used to identify the wallet and its associated bitcoins, while the private key is used to sign transactions and control access to the bitcoins stored in the wallet.

To connect to the Bitcoin network and validate transactions, the wallet uses a remote procedure call (RPC) interface to communicate with the Bitcoin Core node. The wallet sends transactions to the node, which then broadcasts them to the network for confirmation and inclusion in the blockchain. The wallet also receives updates from the node about the status of transactions and the current balance of bitcoins in the wallet.

The connection between the wallet and the node is managed by the connection manager component of the architecture, which ensures that the wallet is always connected to a healthy and reliable node. The connection manager can detect if the current node becomes unavailable or slow and will switch to a different node if necessary to ensure that the wallet can communicate with the network effectively. This helps to ensure that the wallet is always able to send and receive transactions, even if one node fails.

## Validation Engine

The *Validation Engine* in Bitcoin Core is responsible for maintaining the security and integrity of the Bitcoin network. The engine performs a series of checks on transactions and blocks to verify their validity and enforce the consensus rules of the network. These checks include:

1. Verifying transactions: The validation engine checks that each transaction is valid and follows the rules set by the Bitcoin network, such as ensuring that the inputs of a transaction are unspent and that the sum of inputs is greater than the sum of outputs.
2. Checking block structure: The engine validates the structure of each block to ensure that it conforms to the rules set by the network, such as block size limits and difficulty requirements.
3. Protecting against double-spending: The engine prevents double-spending by checking that each transaction only spends inputs that have not been spent before.
4. Validating the blockchain: The engine validates the entire blockchain by following the links from one block to the next and checking that each block is valid.

## Mempool

The data that is being stored on the *Mempool* are the set of unconfirmed transactions. These transactions are waiting in the queue to be validated by the *Miner* before being

put into the blockchain. *Mempool* will use *Validation Engine* to verify the incoming transactions from the P2P network before putting it on the queue.

## Storage Engine

The storage engine component of the Bitcoin Core architecture is responsible for managing the permanent storage of all the data that is processed and validated by the Bitcoin Core node (can be thought of as an API). This data includes the entire history of Bitcoin transactions, also known as the blockchain, as well as other metadata, such as information about blocks, transactions, and addresses. The storage engine component accesses three main databases: Headers, Blocks, and Coins.

The **Headers database** stores block headers with vital information for block validation, such as block hash, previous block hash, and timestamp. Nodes can quickly verify new blocks by checking headers against this database, hence improving performance by avoiding the need to download entire block data.

The **Blocks database** stores block data in serialized binary form. When a node receives a new block, it looks up its header in the headers database and then retrieves full block contents from the Blocks database using the block hash as a key. The node is then able to validate the block and updates its view of the blockchain.

The **Coins Database** stores information about UTXOs, the unspent transaction outputs that can be spent in transactions. It tracks the bitcoin balance of each address and helps validate transactions by ensuring their inputs are unspent UTXOs. To access the database, the query accepts transaction hash and output index as parameters.

## Miner

The *Miner* subsystem tries to find an input to a cryptographic hash function such that the output is less than the **target hash**. Finding a valid input is called “solving the hash” and it does so by incrementing the **nonce** until a satisfying input is found.

The input to the cryptographic hash function includes the list of transactions that are awaiting to be put in the next block of the blockchain, the hash of the latest block in the blockchain, and the nonce. Therefore, *Miner* depends on the *Mempool* subsystem to access the list of transactions as well as the *Validation Engine* subsystem to gain access to the hash.

The verification of the input value is accomplished by having the *Miner* passing in the input value through *Validation Engine*'s validation system.

### Remote Procedure Call

The RPC interface provides a set of commands that can be used by external applications to interact with the Bitcoin Core program. The interface allows execution of certain tasks such as making transactions, as well as getting the balance of the wallet.

## External Interfaces

There are two external interfaces in this architecture.

The first one is the **peer-to-peer (P2P) network**. The P2P network is responsible for connecting full nodes and allowing them to exchange information, such as new transactions and blocks. This helps ensure that all nodes in the network have the same copy of the blockchain and allows for the decentralized and secure recording of information on the blockchain.

The second one is the **external apps and GUI**. These apps are connected to the whole architecture through remote procedure call (RPC). They can use the RPC interface to send commands to the full node and retrieve information such as querying the blockchain for transaction data or sending a transaction to the network. And users can use GUI to get an intuitive view of all the app's interfaces, functions, etc.

## Use Cases

This section discusses two use cases of Bitcoin Core: the process of verifying and recording new bitcoin transactions (known as bitcoin mining), and the process of creating transactions by invoking a remote procedure call.

### Use Case #1: Mining Bitcoins with Bitcoin Core

Bitcoin Core allows the mining of Bitcoins through the built-in console window. The mining process begins when the user enters a command that starts the *Miner*.

When *Miner* is activated, it first asks *Mempool* and *Validation Engine* for the list of unconfirmed transactions in the queue and the parent hash, respectively. The nonce is then initialized to 0.

Using the acquired pieces of data, *Miner* combines them into a single message that resembles the input to the cryptographic hash function that is in the Validation *Engine.* The input string is passed into *Validation Engine* and then waits for a success or a failure notification depending on whether the input solves the hash or not. If the verification is unsuccessful, *Miner* increments the nonce and attempts the check again.

If the input string solves the hash, then *Validation Engine* notifies *Miner* that the hash has been solved and *Miner* proceeds to grab the next set of unconfirmed transactions from *Mempool*, ultimately repeating the same process.

*Validation Engine* also updates the blockchain maintained by *Storage Engine*. The event of updating the blockchain notifies *Connection Manager* to prepare broadcasting of the change to the P2P network. The actual broadcasting is handled by the *Peer Discovery* subsystem which queries for known peers through the *Peers* database and broadcasts the blockchain update to the P2P network.

If a peer in the P2P network solves the hash before the user, then *Validation Engine* notifies *Miner* to cancel the work and start working on the next set of unconfirmed transactions maintained by *Mempool*. The *Validation Engine* also tells *Storage Engine* to update the database.

If the user tells the program to stop mining, *Miner* stops mining.

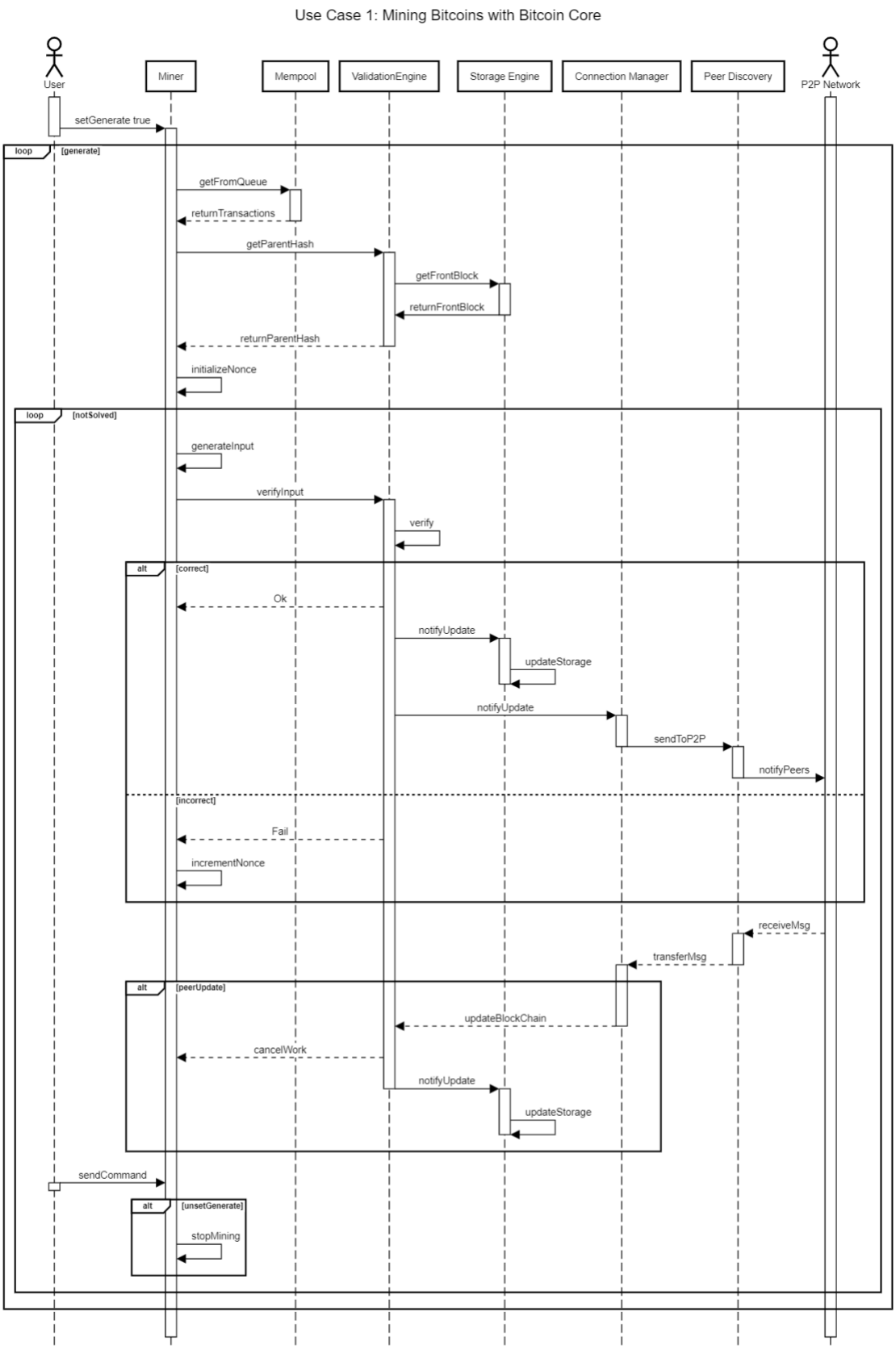


*Figure 2: Sequence diagram for Use Case 1*

## Use Case #2: Creating transactions using a Remote Procedure Call

Bitcoin Core allows external programs to make transactions through the RPC interface. The external program first asks for a request from the RPC interface. Then, a message is sent to the Wallet. To check the balance, the Wallet contacts the Storage Engine.

After getting the balance from the Storage Engine, Bitcoin Core detects whether the user has enough Bitcoins to make the transaction. If the user has enough Bitcoins, the transaction is sent through the Connection Manager.

After that, the Connection Manager performs two actions. On one hand, the Connection Manager goes to the Mempool to add the transaction to the queue. On the other hand, *Connection Manager* tells Peer Discovery to notify the P2P network that a transaction has been made.

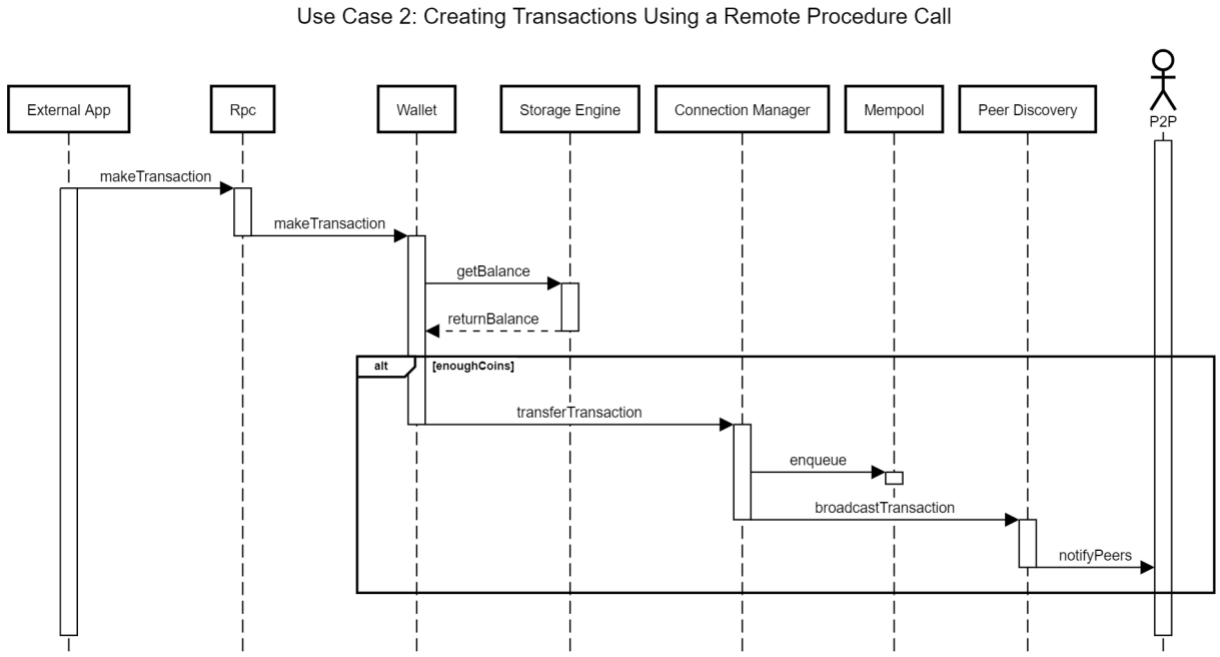


*Figure 3: Sequence diagram for Use Case 2*

# Data Dictionary

**Target hash** - numeric value that *Miner* attempts to find a hash that is less or equal to it

**Nonce** - 32-bit value that is adjusted by *Miner* to solve the hash
**Public key** - cryptographic code that's paired to a private key, which allows you to receive bitcoin transactions
**Private key** - cryptographic code that enables the user to make a Bitcoin transaction and prove ownership of their holdings
**Blockchain** - a decentralized ledger of all transactions stored across nodes in the P2P Bitcoin network
**Block** - a set of Bitcoin transactions from a certain time period
**Header** - Portion of a block that contains information on metadata, hashed representations of the block data, and a cryptographic nonce.
**Unspent transaction output** - Bitcoins which have been authorized by one account to be spent by another

## Naming Convention

**App** - Application
**RPC** - Remote procedure call
**API** - Application programming interface
**P2P** - Peer-to-peer
**UTXO** - Unspent transaction output
**CM** - Connection Manager

## Conclusion

The open-source nature of Bitcoin Core made it easier for us to understand and derive its conceptual architecture. We were able to conclude that Bitcoin Core operates using the P2P, Layered, and Client-Server architecture styles. Among them, P2P is the most important part since the whole architecture is P2P style. As for the other two, Layered style is used for the program itself and Client-Server style is used to enable the interaction between external applications and Bitcoin Core. These findings lead to the discovery that a program can utilize both P2P and Client-Server architecture at the same time. We were also able to conclude that the relationship of the Validation Engine to other subsystems makes the program a reliable and secure software platform for managing Bitcoin transactions.

## Lessons Learned

Over the course of the project, we have faced many challenges and learned many lessons. One major lesson is the importance of streamlining our ideas with a weekly meeting before our major writeup. In this assignment, different ideas about the architecture have made many parts of the report inconsistent and resulted in much redone work. We also saw the importance of knowing the material well before the meeting. Some team members did not do their research and as a result they did not contribute much to the meeting. In the future, we will always streamline the main ideas in our meetings and enforce research across all members.

Another category of lessons are the ones dealing with the analysis of the architecture and breaking it down into parts. We have learned that having a foundational diagram like the one in the open-source handbook is very important to start visualizing the system and all its relations. In the future we hope to start all our research on possibly a diagram that outlines many different parts of the system.

## References


### Original Diagram

[1] Antonopoulos, A. M. *Chapter 3: 'bitcoin core: The reference implementation'*. GitBook. Retrieved February 10, 2023, from https://cypherpunks-core.github.io/bitcoinbook/ch03.html

### Miner

*Mining with Bitcoin Core*. Packt. Retrieved February 10, 2023, from https://subscription.packtpub.com/book/hardware-and-creative/9781785281976/2/ch02lvl1sec12/mining-with-bitcoin-core

### Peer Discovery

Antonopoulos, A. M. M. *Chapter 6: The Bitcoin Network*. O'Reilly Online Learning. Retrieved February 10, 2023, from https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch06.html

*P2P network*. bitcoindeveloper. Retrieved February 10, 2023, from https://developer.bitcoin.org/devguide/p2p_network.html

*Bitcoin Core 0.11 (CH 4): P2p network*. Bitcoin Core 0.11 (ch 4): P2P Network - Bitcoin Wiki. Retrieved February 10, 2023, from https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_4):_P2P_Network

## Mempool

*Mempool Size (Bytes)*. Blockchain.com. Retrieved February 10, 2023, from https://www.blockchain.com/explorer/charts/mempool-size

Coinguides. (2021, August 28). *What is Bitcoin Mempool? memory pool size, Fees & Transactions explained*. Coin Guides. Retrieved February 12, 2023, from https://coinguides.org/bitcoin-mempool/

## Wallet

*Bitcoin Core*. Encyclopedia. Retrieved February 12, 2023, from https://encyclopedia.pub/entry/33603

Coinbase. *Crypto basics - what is a crypto wallet?* Coinbase. Retrieved February 13, 2023, from https://www.coinbase.com/learn/crypto-basics/what-is-a-crypto-wallet

## Storage Engine

Josh. (2021, March 26). *Where are unspent transaction outputs (UTXOs) stored?* CryptoChamp. Retrieved February 13, 2023, from https://cryptochamp.com/where-are-unspent-transaction-outputs-utxo-stored/

Zhang, C. *A Blockchain Network node*. ResearchGate. Retrieved February 14, 2023, from https://www.researchgate.net/figure/A-Blockchain-Network-node-A-full-node-stores-all-the-data-in-the-blockchain-including_fig1_329464413

*Eli5: Blockchain explained in simple terms*. crypto.bi. Retrieved February 13, 2023, from https://crypto.bi/eli5-blockchain/

## Validation Engine

(2019, May 5). *Transaction verification engine*. Stack Exchange. Retrieved February 14, 2023, from https://bitcoin.stackexchange.com/questions/87487/transaction-verification-engineAntonopoulos, A. M. *Chapter 11: 'Bitcoin Security'*. GitBook. Retrieved February 10, 2023, from https://cypherpunks-core.github.io/bitcoinbook/ch11.html