CISC 322
Assignment 2: Report
Concrete Architecture of Bitcoin Core
March 23, 2023

Team BigTime

Azeem Khan (Presenter 1) - 19aik2@queensu.ca
Ben Tomkinson (Presenter 2) - 17bat3@queensu.ca
Lucas Wong (Team Lead) - 20lylw@queensu.ca
Oscar San - 19os14@queensu.ca
Kenny Wong - 20klkw@queensu.ca
Yannik Brunzema - 19ycb@queensu.ca

Table of Contents

Abstract

Introduction and Overview

Top level Concrete Architecture

- Recap of Conceptual Architecture
- Derivation Process
- Concrete Architecture
- Description of Subsystems & Interactions
- Reflexion Analysis

Subsystem Analysis: Mining

- Inner Architecture Analysis
- Reflexion Analysis: Mining Subsystem

Sequence Diagram

Naming Conventions

Conclusion

Lessons Learned & Team Issues

References

Abstract

This report presents a focus on the creation and analysis of the group's concrete architecture for Bitcoin Core, the Bitcoin network's open-source software implementation allowing safe transactions to occur between users without the requirement of a third party.

In our previous report, the conceptual architecture of the Bitcoin Core P2P payment processing system was analyzed. Following a brief summary of our proposed architecture, the concrete architecture of the mining subsystem was analyzed using the Understand tool. Afterwards, a reflexion analysis was performed to highlight key discrepancies between the proposed conceptual architecture, and the concrete architecture. Upon completion of the analysis, the mining subsystem was then examined in more detail. Lastly, we discussed the team's overall issues regarding the tool Understand and time management while reflecting upon the numerous lessons learned from overcoming these problems.

Introduction And Overview

Bitcoin Core, created in 2009 by Satoshi Nakamoto is the open-source software implementation of the Bitcoin network, a decentralized digital currency system that allows for peer-to-peer transactions without the need for a middleman or bank service. The conceptual architecture of the Bitcoin Core is designed with the purpose of conducting highly secure transactions over a stable network.

In this report, we explore the concrete architecture of the Bitcoin Core system, by recovering the empirically observable components through the use of the Understand software to analyze the Bitcoin Core system, and extract the dependencies between its various subcomponents, from which a reflexion analysis is then performed to understand the rationale along with the differences between the concrete and conceptual architectures.

Top Level Architecture

Recap of Conceptual Architecture

Our group had originally proposed that the Bitcoin Core system was represented by a P2P style architecture. Refer to figure 1 below.

Derivation Process

To derive the concrete architecture of Bitcoin Core, we analyzed the static dependencies in the source code between various source code files that are contained within different submodules in the Bitcoin Core architecture. Then, the P2P network interactions were analyzed in the same way as well to accurately derive more of the concrete architecture while gaining further insight into its dependencies. To illustrate, the file key.cpp, depends on the following other source files.

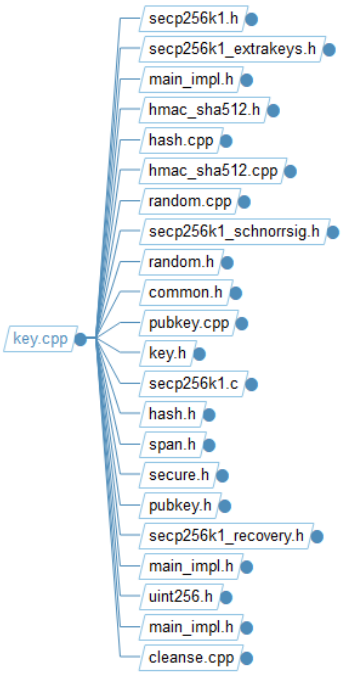


Figure 1. The Dependencies Of key.cpp

As we can see, the key.cpp file depends on various other source files, including hash.cpp, which would imply that there is a dependency between the wallet and security components, as hashing is needed for the keys.

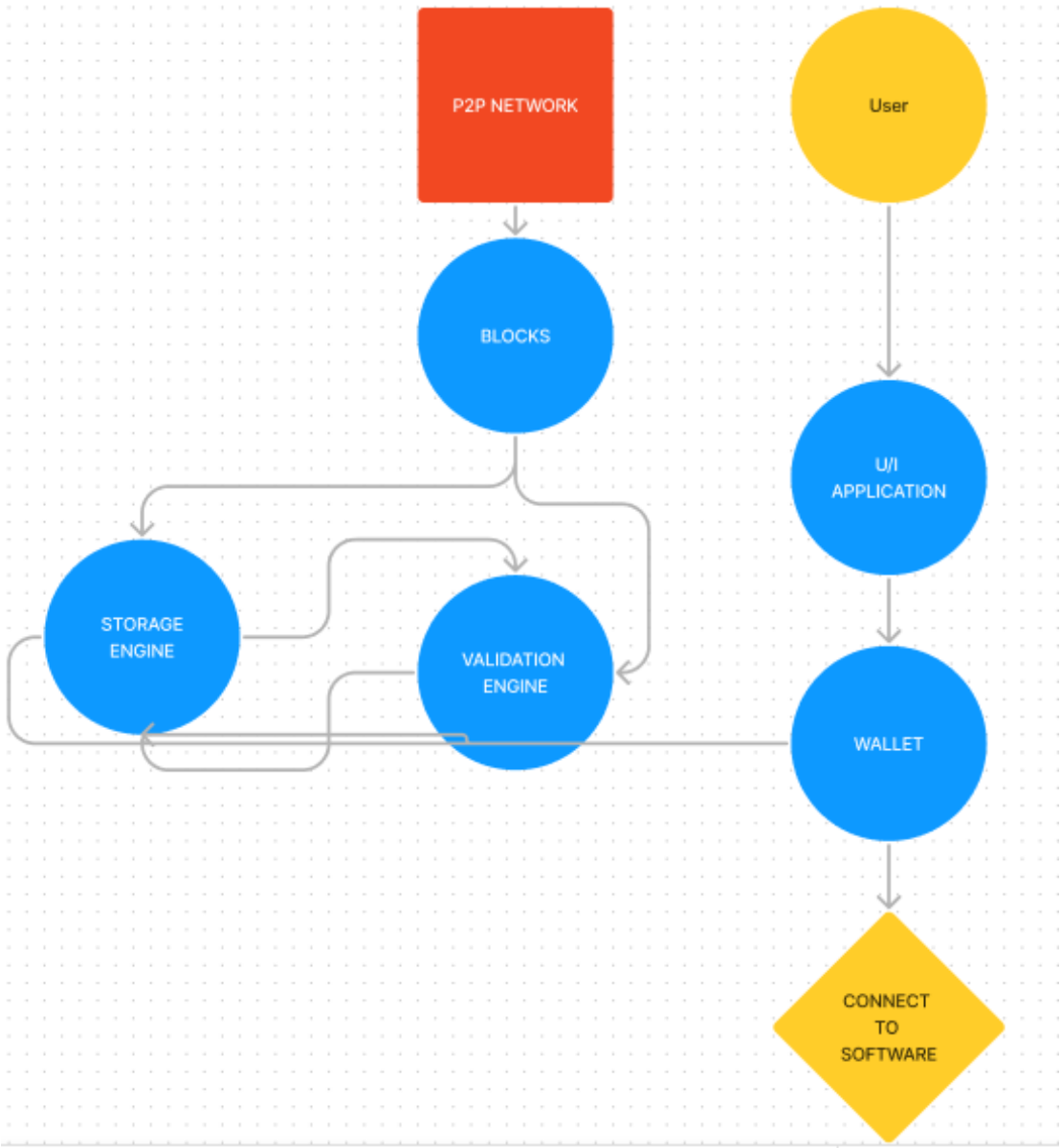


Figure 2: The Conceptual Architecture Proposed In A1

The user interacts with the UI and makes connections to the wallet which then connects to the software of bitcoin core. The wallet obtains its core functionality through its communication with the storage and validation engines which are dependent on each other. These two engines are closely connected to the P2P network, indirectly creating an interaction between the wallet and the network's branching components which is essential to the overall architecture of the software.

Concrete Architecture

Based on the above derivation process, which combines static dependencies, as well as the analysis of the P2P network interactions, we arrived at the following concrete architecture seen in Figure 3 below.

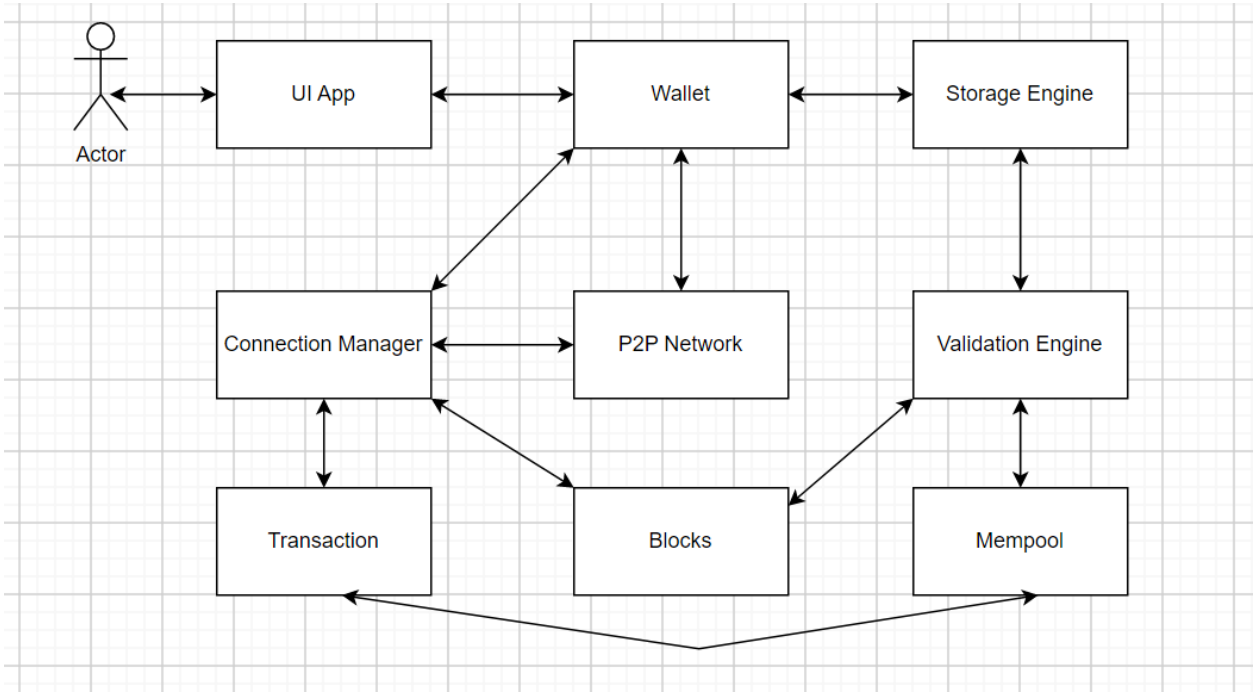


Figure 3. Bitcoin Core Concrete Architecture

Description Of Subsystems & Interactions

UI App: The UI interface that the user interacts with. There is a mutual dependency between the wallet component and the wallet subsystem.

Wallet: The wallet is the main way a user interacts with bitcoin, it controls access to their currency, managing the keys and addresses while tracking the balance of the user's wallet. The wallet subcomponent has a mutual dependency with both the UI App subsystem, as mentioned above, as well as with the storage engine component, due to the fact that the storage engine stores and retrieves the necessary blockchain data required for the transactions to take place.

Storage Engine: Responsible for managing the storage and retrieval of the blockchain data. The storage engine functions implement the data structures and algorithms required to efficiently store and retrieve blocks and transactions. It interacts with the validation engine as it stores validated transactions, as well as the wallet so it can send the relevant user information to the wallet.

Connection Manager: The connection manager is responsible for holding the list of connected peers as well as maintaining a healthy connection with other peers. Because many subsystems require a connection with peers to function, many depend on the connection manager, such as the transactions, the blocks, and the wallet. Also, the manager accesses the actual P2P network to be able to instantiate a connection between peers.

P2P Network: The P2P network is the main architectural style of the bitcoin system. It enables decentralized communication and transactions as each node in the network is connected while directly communicating and sending transactions to other nodes. It interacts with the connection manager as it sends it a list of connected peers, as well as the wallet because users instantiate a transaction from the wallet, which it then has to propagate to the network.

Validation Engine: The validation engine functions check the validity of transactions and blocks to ensure that they meet the consensus rules set by the bitcoin protocol. The engine verifies each transaction's signature and its spending limits, as well as proof-of-work, transaction validity, and difficulty target of each block. The validity engine interacts with blocks, feeding it approved transactions and the storage engine, to store data.

Transaction: A transaction is a set of data structures that encode the transfer of value between connected peers. These transactions are what allow bitcoin to be used as a form of currency, therefore a strong validation system is necessary to ensure safe transfer of bitcoin. It interacts with both the connection manager, to only connect with peers in the network, and the mempool, which is the storage of unconfirmed transactions that await to be validated.

Blocks: A block is a set of data structures that stores a set of confirmed transactions, with each block containing a hash that leads to a previous block creating a chain. The point of these blocks is to create an archive of previous transactions, and to add security to these transactions. Because the blocks are linked together, you cannot change one without changing the other which adds a layer of protection. These blocks interact with the validation engine as it receives a validated transaction from it to be accepted into a block. It also interacts with the connection manager since the blocks would only like to receive data from connected peers.

Mempool: The mempool is a data structure that stores data regarding unconfirmed transactions waiting to be confirmed, where the confirmed transaction can then be accepted into a block. Therefore, it has to interact with the transactions system, to store a transaction everytime one occurs, and the validation engine, which is used to validate a transaction.

Reflexion analysis

As can be seen from the concrete architecture above, the peer to peer network architecture remains consistent with the conceptual architecture. However, new nodes and dependencies are added to give greater insight on the specifics while showing functionality and implementation.

Discrepancy: New nodes have been added in the concrete architecture. Namely the transaction node, mempool node, and the connection manager node.

Rationale: These new nodes are implemented to show functionality and an idea of how other nodes can be implemented. The transaction node is what allows users to transfer bitcoin values from one address to another. The mempool node allows for the storage of unvalidated transactions to be waited on, and to be accepted into a block. Lastly, the connection manager node works in tandem with the P2P network and is responsible for getting and maintaining access to other users on the network.

Discrepancy: In the conceptual architecture, the wallet is dependent on the storage engine and the UI software. The concrete architecture has the wallet dependent on the UI, storage engine, P2P network, and the connection manager.

Rationale: The wallet is dependent on the connection manager and the P2P network, this is to allow users to control coins on the network by signing transactions with the keys in their wallet. The wallet is also dependent on the UI and storage engine because of its primary purpose to act as a user interface.

Discrepancy: Originally in the conceptual architecture, blocks were directly connected to the P2P network. In the concrete architecture, the blocks node is now connected to the connection manager

Rationale: The connection manager holds a list of connected peers, having the block node connect to the manager instead of straight to the P2P network better reflects how our system would implement such a function, as bitcoin interactions cannot happen unless peers are connected.

Discrepancy: The transaction node is added because it is crucial to the functionality of the system. Our conceptual architecture had the infrastructure in place to facilitate such a functionality and in the high level concrete architecture the transaction node is here to implement such a function.

Rationale: The transaction function is the most important part of the bitcoin system. It allows data structures that hold the transfer of value between connected peers in the bitcoin system. This node is connected with the connection manager to acquire the list of connected peers. More importantly, it is dependent on the mempool node, as transactions have to be first validated before being allowed into a blockchain to deter false transactions. Therefore, after a transaction, the data first waits in the mempool to be validated before it is allowed into a block.

Subsystem Analysis: Mining

Inner Architecture Analysis

The mining subsystem in Bitcoin Core is responsible for generating new blocks by solving a computationally intensive problem known as Proof of Work (PoW). The miner's g oal is to find a solution to the PoW puzzle, which allows them to propose the next block to be added to the blockchain.

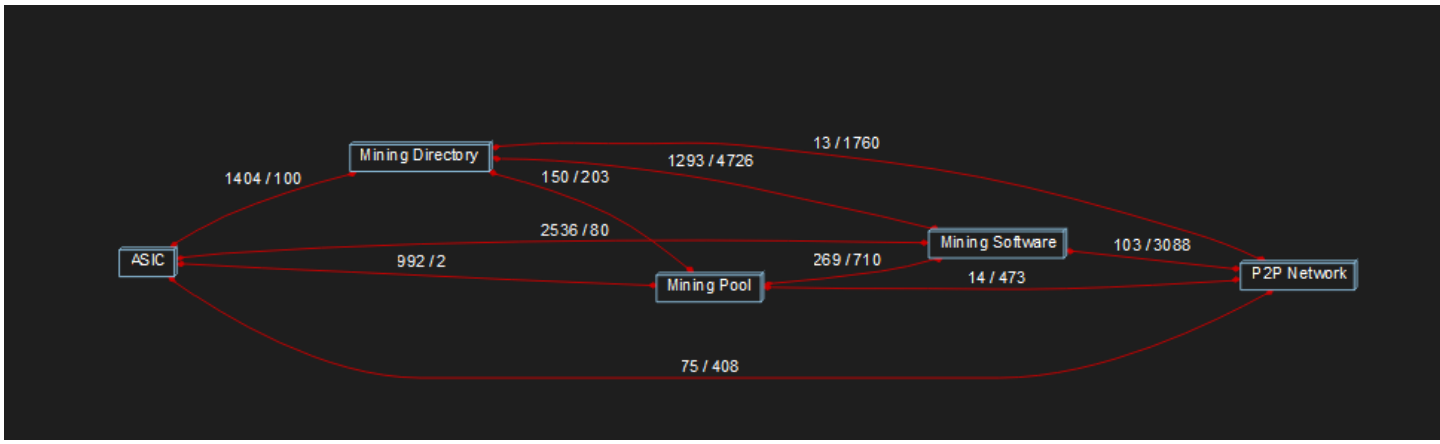


Figure 4: Internal dependencies of Bitcoin Cores mining software modules, graphed using Understand (Build 1135)

Conceptual View:

a. Proof of Work (PoW): Bitcoin mining relies on the SHA-256 hashing algorithm. The mining process involves repeatedly hashing the block header while changing a value called a nonce until the resulting hash is less than the target value. The target value is adjusted based on the network's overall hash rate, aiming to maintain a constant block creation rate (approximately every 10 minutes).
b. Block Template: When mining a new block, the miner creates a block template containing the block's metadata (e.g., version, previous block hash, timestamp, and nonce) and a Merkle tree of pending transactions. These transactions are selected from the memory pool (mempool), prioritizing those with higher fees.
c. Block Propagation: When a miner successfully mines a block, they broadcast it to the network. Other nodes validate the block by checking that the PoW is valid and that the transactions included in the block are valid and not double-spent. If the block is valid, nodes update their local blockchain copy and start mining on top of the new block.

Concrete View:

a. Mining code (software): This part of the program is responsible for creating new block templates and mining new blocks. It also handles the PoW algorithm, which is a complex puzzle miners need to solve in order to create a new block. The software is essential to handling transactions, as well as assisting the validation of new blocks and transactions.
b. Block and transaction validation: This component checks whether a block or transaction follows the rules of the Bitcoin network, ensuring that everything is valid and secure.

c. Mempool: The mempool stores unconfirmed transactions until they are included in a block. It also helps miners choose which transactions to include in a new block based on the fees they offer.
d. P2P networking: This part of the program manages communication between computers on the Bitcoin network, allowing them to share blocks and transactions with each other.
e. ASIC: ASICs (Application-Specific Integrated Circuits) are designed and built for the use of mining. After transactions are validated, and all the information is passed through the mining pool in the software, it is run on the ASIC hardware.
f. Mining Directory: The mining directory acts as a shared folder that contains files and information that is essential to all parts of the subsystem. This includes files related to IPC (inter-process communication), ZMQ (ZeroMQ), general policies and more.
g. Mining Pool: A mining pool is a workflow similar to solo mining, that contributes computational power to solve hashes, and distributes rewards among the group. They routinely get new transactions from the network, using *bitcoin* and various other methods.

By conducting an in-depth review of Bitcoin Cores' subsystem of Mining, we were able to gather a basic understanding of its modules, relationships and dependencies. We were able to validate the internal dependencies of the architecture using the Understand software tool. Through examining the architecture of the 2nd level subsystem for mining, we gained a greater understanding of the dependencies that lie in the subsystem, such as between the mining software, and the ASIC hardware. In terms of architectural styles used, the subsystem remains to use a peer-to-peer (P2P) architecture for communication between nodes, although in terms of alternative architectural styles, the mining subsystem is built around using a client-server style, as the mining pool acts as a server-like system to communicate with the designated mining software (client). In terms of structure, the system follows a built-on pattern, with each component interacting with the next, while the mining pool acts as a publisher that receives new transaction information, and distributes it along the network to miners who wish to receive it.

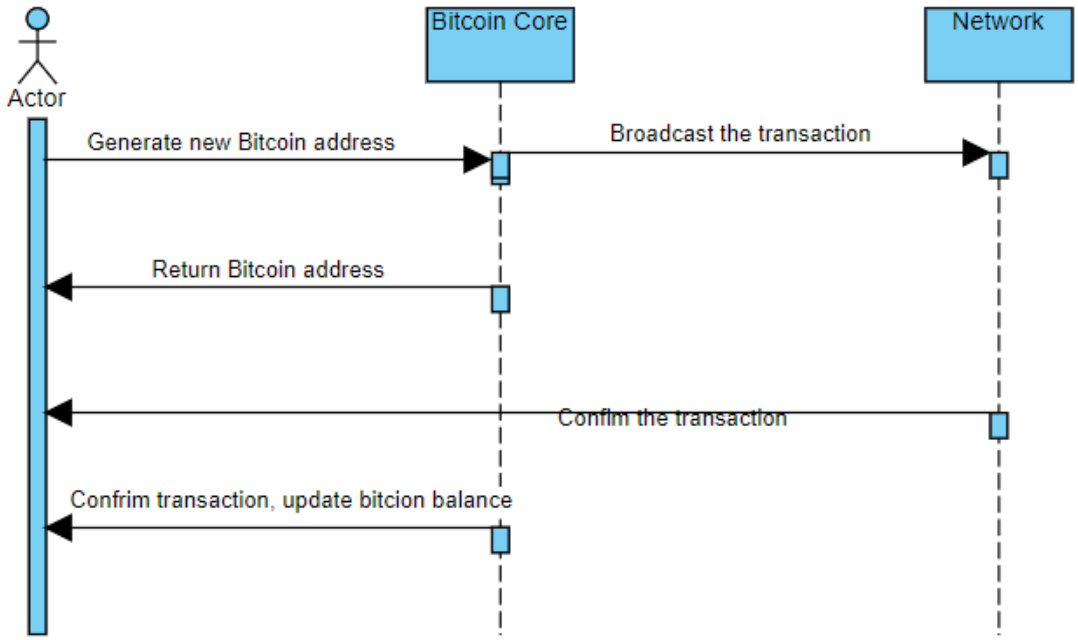
Reflexion Analysis: Mining Subsystem

From the conceptual view: the mining subsystem is responsible for generating new blocks and by solving a Proof of Work. This is done by hashing the block header while changing a value called a nonce until the resulting hash is less than the target value. When mining a new block, the miner creates a block template containing the block's metadata. When a miner successfully mines a block, they broadcast it to the network. Other nodes validate the block by checking that the PoW is valid and that the transactions included in the block are valid and not double-spent.

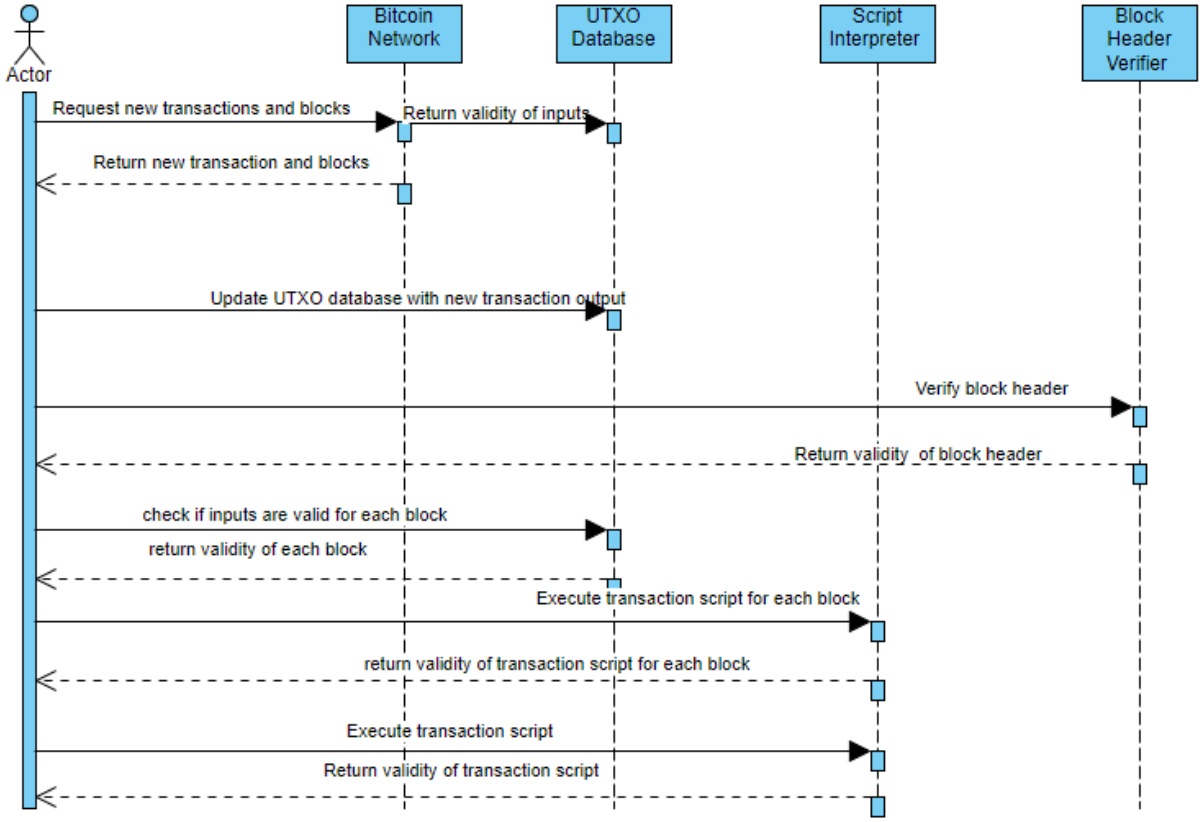
From a concrete view: Mining code is responsible for creating new blocks and handles the PoW to create new blocks. Block and Transaction validation, checks if a block follows the rules of the Bitcoin network. Mempool, stores unconfirmed transactions. P2P, manages the connections between computers on the network, the mining pool contains and transfers transaction information and solved hashes, while ASICs handle real-time mining.

Sequence Diagram

Storing Value:



Block validation:



Naming Conventions

Peer-to-Peer Network (P2P): A decentralized communications model where either party initiates a communication session, with both parties having the same capabilities.

Dot C Plus Plus (.cpp): A file extension for the C++ programming language, used to create and program high-performance applications.

UI (User Interface): The means of which the user is able to interact with the systems involved with the software.

ASIC (Application-Specific Integrated Circuits): Large-scale computer chips that combine several different circuits, in order to carry out a specific task. Used prominently in mining, and uses PoW (Proof of Work) algorithms.

IPC (Inter-process Communication): Used for programs to communicate data with each other, and synchronize activities. Common methods for IPC include queues, shared memory and semaphores.

I2P (Invisible Internet Project):

ZMQ (ZeroMQ): An asynchronous messaging library used in distributed or concurrent applications. ZMQ is able to be run without the need for a dedicated messaging broker.

Conclusion

The team was able to derive the concrete architecture, its subsystems and how they interact together, using Understand and help from the Bitcoin core Github. We explored the discrepancies between the conceptual architecture and the concrete architecture and performed reflection analysis. This resulted in us seeing just how much Bitcoin Core changed from its initial inception to its concrete architecture. Then we took a deeper look into one of Bitcoins core specific subsystems and looked at the dependencies using Understand. This highlighted how much every subsystem is interconnected. Then, the team came up with sequence diagrams for two non-trivial use cases; these were for storing value on the block chain and for block validation. Overall, we finished with a well defined outlook on how the architecture changed from its conceptual self and got a look into its functionality and domain.

Lessons Learned & Team Issues

During the duration of this project, our team came across many different issues that we were able to solve together. These issues pertained to the use of Understand, decoding the documentation. We struggled with using Understand for the creation of and look into the inner architecture of the subsystem due to the amount of information and files within Bitcoin Core. We did not properly estimate the time this task would take and it definitely resulted in a lot of stress on the team to resolve this deliverable. Going forward, budgeting our time better will be key to creating a steady pace for the team's work cycle.

Upon reflection of our time budgeting, team scheduling was also an apparent issue this time around as everyone had many various commitments. This led to large amounts of independent work time resulting in work redundancy and some tasks being ignored which caused a lot of last minute confusion. While this was a big issue, it was the team's collective fault and we all think we can do better.

On the more technical side, we struggled to produce dependency graphs for the subsystem of mining due to vague documentation on the Bitcoin core github which left much of

the necessary systems that needed to be included to be inferred by us. This leads to one of the biggest obstacles we faced during this project, was our limited knowledge of the Understand software. While the software was instrumental in the creation of our graph and its dependencies, we found that not only did it frequently crash when met with too many requests, there was only so much information available regarding the dependencies. This led us to analyzing the raw source code of the subsystem, which was difficult when some of us were unfamiliar with the code, syntax and meaning behind it. We truly were unsure at many points in the creation of it, but are confident in our effort and hard work.

Finally, due to our poor conceptual architecture representation, it made the process of reflection analysis exceedingly, and unnecessarily, difficult.

References

Bitcoin. (n.d.). *Bitcoin Core Source Code Repository*. GitHub. Retrieved February 18, 2023, from https://github.com/bitcoin/bitcoin

Antonopoulos, A. (n.d.). *Mastering Bitcoin*. Introduction · GitBook. Retrieved February 18, 2023, from https://cypherpunks-core.github.io/bitcoinbook/

Developer Guides. Bitcoin. (n.d.). Retrieved February 18, 2023, from https://developer.bitcoin.org/devguide/index.html#

Nakamoto, S. (n.d.). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Bitcoin. Retrieved February 18, 2023, from https://bitcoin.org/bitcoin.pdf%20

Investopedia. (n.d.). Investopedia. Retrieved February 19, 2023, from https://www.investopedia.com/

Understand: An IDE and static code analysis tool by SciTools. (n.d.). Retrieved March 24, 2023, from https://www.scitools.com/