Architectural Enhancement of Bitcoin Core

Authors

Devin Pereira, Kanchan Shrestha, Alex Susilo, Adlai Bridson-Boyczuk, Cyrus Fung, Isaiah Wuthrich

Abstract

In this third and final report, we propose a new feature to be added onto the existing Bitcoin Core architecture. The feature we are proposing as an enhancement is an API for currency-pair and exchange rate information. We begin this report with two different methods this new feature could be implemented, and a SEI SAAM analysis covering both the aforementioned approaches. Followed by an in-depth look at the impact of this feature on current subsystems in the architecture, potential risks, and limitations of implementing this feature, and a layout of testing that will be done to assess the impact or changes that have happened. Beyond this we will use two use cases in order to show the utility of a currency and exchange rate API: (i) bitcoin transaction involving currency exchange rates, and (ii) node updating its exchange rate from a peer. Finally, we conclude with a summary of the overall report and the lessons learned as a result of planning this new feature.

Introduction

As technology rapidly advances, Bitcoin remains at the center of attention being one of the most well-known and widely adopted cryptocurrencies. Bitcoin Core, the reference implementation of the Bitcoin Protocol, is open-source software that works as the backbone of the Bitcoin network. Its responsibilities include validating transactions, mining blocks, and maintaining the integrity of the blockchain.

So far, our previous reports have entailed an in-depth analysis and comparison of the conceptual and concrete architectures of the Bitcoin Core software. Now, our team will be introducing a new feature: an API for currency pair and exchange rate information. In recent years, many newer cryptocurrencies like Ethereum (ETH), and Solana (SOL) have begun storing exchange data on the blockchain, so why doesn't Bitcoin? Obtaining an accurate representation of this information is incredibly important for any transaction that may occur on the network and yet it does not exist for the world's largest cryptocurrency.

In this report, we will provide an overview of the proposed API enhancement for Bitcoin Core, followed by a comprehensive discussion of two different approaches to implementing this enhancement: a peer-

to-peer sharing based system and a more traditional client-server-based API. We will then conduct a thorough SEI SAAM analysis, which includes identifying stakeholders, defining non-functional requirements, and assessing potential impacts on these requirements. Furthermore, we will delve deeper into the potential impacts on the underlying subsystems and the overall concrete architecture of Bitcoin Core. This will be followed by a discussion on potential risks and security limitations associated with the proposed enhancement. Additionally, we will illustrate two new use cases involving this feature using sequence diagrams. Next, we will discuss various testing methodologies, including unit and integration testing, to ensure the robustness and reliability of the system with the new feature. Finally, we will share the lessons learned from writing this report and provide a concluding summary of our findings.

Overview of New Feature

As cryptocurrency gains popularity, an increasing number of people are interested in investing in digital coins like Bitcoin. However, buying and selling Bitcoin can be convoluted, especially for those new trading. Our team acknowledges this challenge and proposes a new feature for the Bitcoin Core app that aims to simplify the process. This feature offers real-time exchange rate information for Bitcoin in the users preferred currency, all within the app. This exchange rate information is crucial for trading Bitcoin as it allows users to make informed decisions about when to buy or sell. With this information accessible within the app, users no longer need to check external platforms for the current exchange rate. Additionally, the new feature allows users to set their own personal exchange rate for Bitcoin, giving them greater control over their investments and potentially increasing profits. Ultimately, we believe this feature will enhance the user experience of the Bitcoin Core app and facilitate wider adoption of cryptocurrency by streamlining the trading process, promoting efficiency, and increasing transparency.

Approach 1

The first approach involves the inception of a new subcomponent named Rates which would fit within Storage Engine. The Rates subcomponent, present in every node, would be responsible for storing exchange rates from Bitcoin to other currencies as per its name. Nodes could set these rates in a variety of ways.

The first method which could be implemented would be for nodes to set their own exchange rates to indicate the price at which they are willing to trade their Bitcoin. This service could be implemented as a simple addition to the GUI: a field in which a user would enter their desired rate and a button for validation. From there the data would be sent from the App component to the Storage Engine. Data might be made to flow in the opposing direction for the current exchange rate to be displayed in the GUI. Naturally, this functionality would be supported by the functions of the CLI.

There is another way of obtaining exchanges rates which could be implemented in parallel. Nodes could clone rates from another peer. Users would initiate this process from the App component, calling the Connection Manager component to contact other peers. The selected peer's Connection Manager would obtain the exchange rate data from the Storage Engine and would communicate it to the original node's Storage Engine by way of its Connection Manager. A derivative feature could be implemented to synchronize a node's exchange rate with the average rate across the network. In an equilibrium situation where all nodes have the same exchange rate, if a peer were to trade their bitcoin for another

currency, their exchange rate would be updated as a result of the transaction. This exchange rate would create a small change, forming part of the average rate which would propagate across the network to each peer.

Approach 2

An alternative to the peer-to-peer approach would be to allow each node to specify its own source for currency and exchange rate data. This approach requires the creation of a new submodule called Exchange API, which would continually make GET requests to a user-specified URI. In most cases, the data source would be a pre-existing cryptocurrency exchange that provides currency and exchange rate data in a standardized format. The Exchange API subcomponent would be responsible for storing URIs and requesting, parsing, and finally validating data received from external APIs. Data would then be passed to other subcomponents to be either displayed to the user (in the App subcomponent) or exposed in Bitcoin Core's own internal API.

SAAM Analysis

Stakeholders

There are two main groups of stakeholders that this enhancement would affect: Bitcoin Core Developers, and Bitcoin Network users. Bitcoin Core Developers despite being a small portion of those impacted are the most impacted and also differ the greatest from the rest of the users which is why we have separated them into their own group. Bitcoin Network Users are divided into five groups: Bitcoin Miners, individuals, or entities that valid transactions and add them to the blockchain; Bitcoin Exchanges and Wallet providers, platforms or services that let users buy, sell, or store their Bitcoin holdings; Bitcoin Merchants, companies or others who accept Bitcoin as a form of payment; General users, any individual who uses Bitcoin for sending and receiving payments; and Financial Institutions, such as organizations or banks that use Bitcoin as a form of investment. In the following we outline NFRs for the primary stakeholders leaving out and merging less impacted stakeholder groups.

NFRs for each stakeholder

Bitcoin Core Developers
The most important Non-Functional requirements for the Bitcoin Core developers are Reliability, Scalability, and Maintainability. There is a clear difference in which approach is ideal, as all three of these NFRs are increased with approach one whereas all three are decreased with approach two. When you rely on a centralized server as the client-server approach does, you are risking failure at a single point, meaning a Peer-to-Peer approach works better for reliability. Scalability is ensured because once again you don't rely on a single server, meaning that you don't have to add more servers or computing power just to manage the network load. And in approach one it is dispersed in a decentralized manner. Finally approach one also increases maintainability because it integrates well with all of the other functions which are primarily peer-to-peer based.

Bitcoin Miners
Bitcoin miners are mainly concerned with Accuracy and Performance. The first prominent issue that

approach one has is that nodes may communicate unreliable data on exchange rates and information. This is because if one node has somehow collected the wrong or incorrect data, it will then share that to other nodes propagating the data further. Since approach two does not rely on other nodes and gets the information directly from a centralized server, this is less likely to occur increasing accuracy. As for performance, both approaches require additional overhead to fetch the exchange rate information (Approach 2: GET requests, Approach 1: Propagating and Synchronizing data). For this reason, there is no preference for which approach will better serve the needs of this NFR.

General Users & Financial Institutions

For the general public using Bitcoin Core, Compliance, Security, and Accuracy and Ease of Use rank the highest for Non-functional requirements. It's also worth mentioning that we combined this group as they essentially have the same values since the majority of general users and financial institutions use Bitcoin either as a form of investment or as a means to pay for goods and services. The peer-to-peer approach would make compliance entirely variable based on how the specific user or institution utilizes the software. For this reason, approach two is ideal for compliance. Security will be far more likely to be ensured with the peer-to-peer approach as the decentralized nature of Bitcoin Core is naturally secure, whereas a client-server approach would require additional security measures such as authentication, authorization, and encryption. Accuracy was previously mentioned under Bitcoin Miners and the sample logic applies here, therefore, approach two is ideal. Finally, when considering ease of use it's important to note that most users or institutions would need to rely on their understanding of the Bitcoin Core app if approach one was to be used. In comparison approach two allows for the creation of a user-friendly interface combined with the API.

Architecture Choice

After reviewing and contrasting the two approaches, our team decided that we should implement the API for currency-pair and exchange rate information using the Peer-to-Peer architecture outlined first. There are several reasons for our team coming to this decision.

To start, the Peer-to-Peer architecture allows more potential for scalability and reliability, both of which are non-functional requirements that are shared by the majority of stakeholders (Developers, Miners, Exchanges and Wallet Providers, and Financial Institutions). Peer-to-Peer meets the requirements of: (i) reliability because it allows direct communication between users without a central server, meaning that there is a much lower chance of one failure causing the whole system to fail; and (ii) scalability because once again without a central server there is no need for one server to bare the load of the entire network preventing any bottlenecks that may arise from a increased userbase.

Second, this architecture is far more easily integrated into our current overall architecture. Bitcoin Core already uses a decentralized Peer-to-Peer network for the majority of its functions meaning that if we add this API, it will require less modification of the overall system in order to function properly. Furthermore, we could make use of preexisting functionalities that are already in use for Peer-to-Peer technologies.

Finally, it allows users to set their own personal exchange rate for Bitcoin, giving users more control over their own finances. This also aligns with the non-functional requirements listed under General Users who prioritize ease of use.

Maintainability, Testability, Evolvability, and Performance

Adding a new feature such as this API is most likely to have an impact on maintainability, as it can introduce new dependencies and increase the code complexity. Furthermore, developers will have to consider new cases for error handling, different and new maintenance for data updates as well as maintaining the overall system accuracy. From this the overall maintainability will decrease as the complexity increases

Testability is the efficiency and effectiveness of testing. This can also be affected by our feature addition as new tests might be necessary to validate functionality of the API. There are several possible situations for new testing related to different currencies, exchange rate updates, and error cases. This means that more testing will be necessary across the board including unit, integration, and end-to-end testing.

Evolvability is the potential for a seamless update for the addition, removal, or modification of new requirements. In adding this feature, we will gain the ability to collect exchange rate data and will have a format for how to implement similar features in the future. This means that other future evolutions of the software may become easier to evolve as abstraction or encapsulation of the API logic can be reused elsewhere, therefore this feature increases the Evolvability of Bitcoin Core.

Finally, Performance refers to how well a system performs in terms of throughput, response time, and deadlines. Although real-time exchange rate data does not require much computational power, especially when compared to the rest of the Bitcoin Core system, it still results in additional overhead (network requests, data processing, and UI updates). Overall, further resulting in decreased performance as it has very little change in actual functionality of Bitcoin Core as a whole.

Impact on Subsystems

Connection Manager

Recall that Connection Manager is responsible for peer discovery, node-to-node communication and synchronization. To receive market data from an external resource, Connection Manager will have an additional connection to the Rates subcomponent in a pub-sub manner, where events are price/market updates. In the case where the node is broadcasting or receiving market data from peers, Connection Manager does not require additional connections to other subsystems, as the pre-existing connection to Wallet in the Application Layer is already sufficient to display the data to the user.

Storage Engine

With the new feature, each node will have to store additional data on the disk. That is why a new subcomponent called Rates was created within the Storage Engine component. Rates is responsible for storing information such as the exchange rate to the user's preferred currency as well as the price the user wishes to trade Bitcoin at.

Wallet

Having this new feature has a small impact on the Wallet subsystem. It is responsible for displaying the user's Bitcoin balance and the value of their transactions in flat currency, so to incorporate the exchange rate API, it would need to communicate with the Rates subcomponent in Storage Engine. It would do this through functional calls, periodically querying the subsystem to retrieve the latest exchange rate data.

Validation Engine

In the case that nodes clone their peers' exchange rate data, the Validation Engine subsystem would need to be modified to incorporate additional validation checks for exchange rate data, ensuring that it meets certain criteria before incorporating it into its calculations. For example, the Validation Engine may need to be updated to include checks for digital signatures and other authenticity checks for exchange rate data, as well as additional security checks to ensure that the data is safe to use. The Validation Engine may also need to be updated to incorporate additional error-handling capabilities, in case the exchange rate data received from other nodes is invalid or corrupted in some way.

App

The creation of the new subcomponent Rates within the Storage Engine sub-system would have a less severe impact on the App sub-system. This is mainly in part of the fact that App already features a bi-directional dependency with the Storage Engine. In terms of the impacts, the App sub-system would need to be slightly modified to incorporate the new exchange rate functionality. This would likely involve updating any functions that interact with the Storage Engine to include exchange rate information, such as the value calculation of the user's preferred currency based on the exchange rate information retrieved. Additionally, any existing UI within the App sub-system that displays or modifies the exchange rates would need to be modified to work with the new Rates subcomponent.

Util

With Util's primary responsibility of providing utility functions that are used across various components of the system, the proposed approach would have little impact on the sub-system. However, there may be some utility functions that would need to be added to support the new Rates subcomponent. For instance, the Util sub-system may provide a function for converting Bitcoin to another currency using the exchange rates stored in said subcomponent. This function could be used by other components, such as App, that need to display exchange rates. In addition, the Util sub-system might also provide functions for validating exchange rates and performing calculations on various exchange rate-related fields such as finding the average exchange rate across the network. While the impact on the sub-system would be relatively small, these added utility functions would be used by other components of the system that rely on this data; allowing Bitcoin Core to continue running as a cohesive system.

Impact on Concrete Architecture

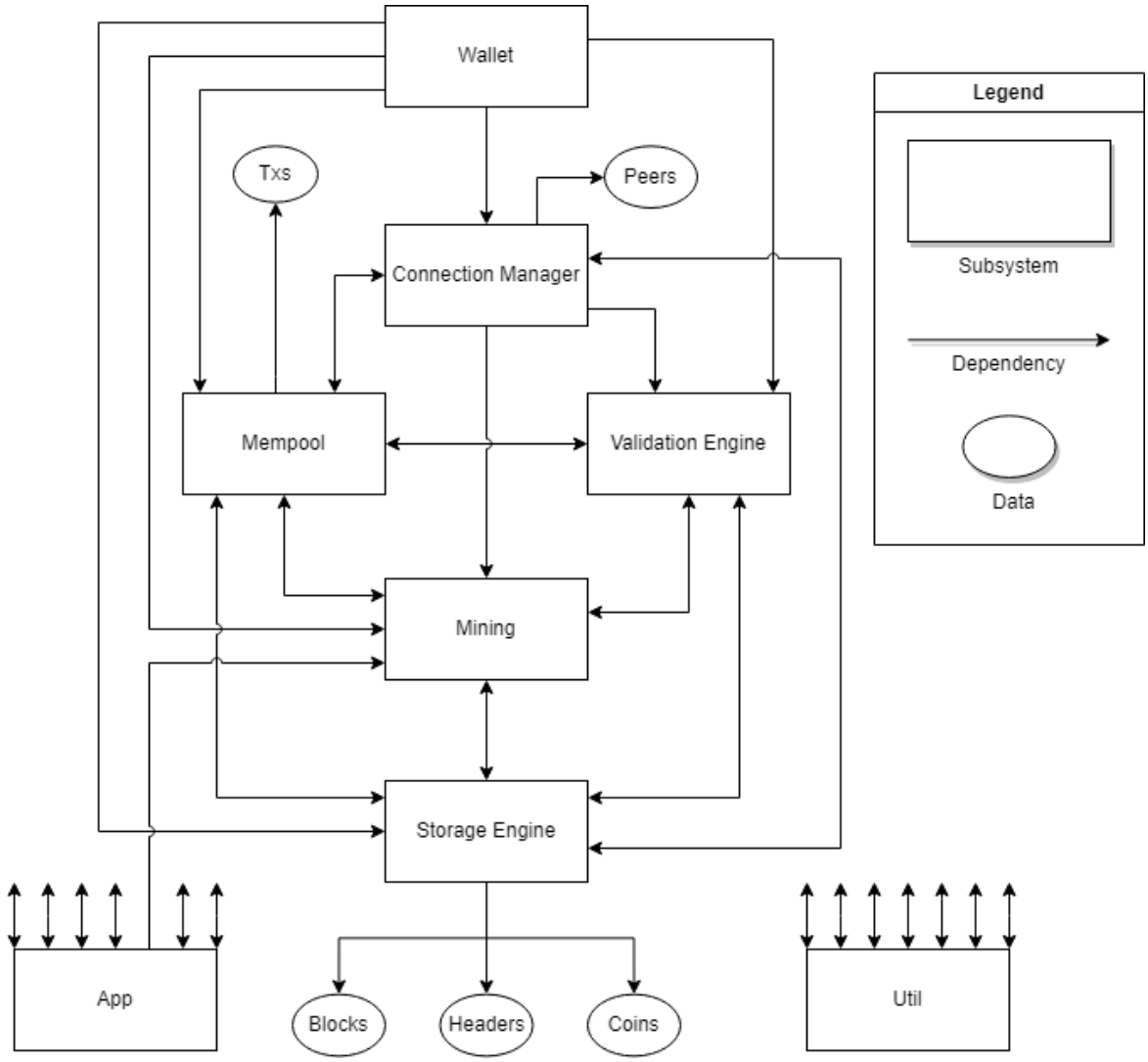

Figure 1. Concrete architecture of Bitcoin Core with the new submodule under Storage Engine

Connection Manager (Cyrus)

A dedicated Market Data Manager submodule would need to be added to Connection Manager to implement the proposed change. This submodule has two configurations. In the first configuration, it obtains market data from the (Rates) module and broadcasts it to other peers using the functionality already included in Connection Manager. In the second configuration, Market Data Manager receives market data from other peers and propagates it through the network.

Storage Engine

The Rates subcomponent within the Storage Engine will store the current Exhange rate to the user's preferred currency as well as the rate they are willing to trade Bitcoin at. As such, the main change will be a new “Rates” folder which will contain files responsible for storing the exchange and trading rates. There are two ways in which the rates can be set then stored on the disk. The first way is when the user

sets the rate themselves and the other is through the Market Data Manager submodule in the Connection Manager, where it will obtain the rate from a connected node. Anytime a node requests the rates of another node, Rates will supply Market Data Manager with the needed information to be sent out.

Wallet

The wallet subsystem will need to communicate with the Storage Engine and retrieve the current exchange rate data. A new file will need to be created containing the functions/ methods that allow for the subsystem to perform these queries. However, no new folders are needed because the only action being done is fetching. Furthermore, no new dependencies are created because Wallet already has a unidirectional dependency on Storage Engine. Overall, the changes to wallet overall are small.

Validation Engine

To implement security checks, a new subcomponent will be added named “checkPeerRate” containing files that check the digital signatures and authenticity check of a peer’s exchange data. This will allow nodes to obtain secure exchange rate data rates from peers before using it to update their own data. This subcomponent will have an incoming dependency from Connection Manager, and an outgoing dependency on Storage Engine. This overall does not change the architecture as these dependencies already existed.

App

Upon designing the new concrete architecture, it is evident that there are no significant impacts on the App sub-system. Compared to the previous concrete architecture without the proposed changes, the App sub-system maintains its bi-directional dependencies amongst all other sub-systems, while having its uni-directional dependency with the Wallet sub-system. App continues to tie together the various components of the software into a cohesive and functional system, with its modular design to ensure that the software can continue to evolve and adapt to meet the changing needs of the Bitcoin ecosystem.

Util

Analyzing the new feature’s influence on the system’s concrete architecture, the Util sub-system also sustains minimal impacts. Inspecting the prior concrete architecture disregarding the new feature, the Util sub-system maintains to have a bi-directional dependency with all the other sub-systems. Of course, seeing how Util depends on all other components to facilitate the common functionality, it is believed that this is a flaw with the system architecture. A utility component should not depend on the components it is designed to service, suggesting that developers may have been cutting corners for the sake of simplicity.

Potential Risks and Limitations Security

Allowing market data to be propagated through the network like transactions creates an entirely new attack surface for Bitcoin Core. Vulnerabilities in the market data parser could allow a malicious node to

exploit other nodes by sending malformed requests. To minimize bugs and vulnerabilities, developers should comprehensively verify all incoming and outgoing market data by restricting datatypes, data length, format, etc. Additionally, to mitigate the impact of any exploits that may be found for the feature, Connection Manager should only export very limited functionality to Market Data Manager. This will significantly increase the level of difficulty of successfully exploiting a vulnerability.

Scalability

Increasing the amount of data being sent on average from peer to peer could lead to serious scalability concerns. Since exchange rate data requires a high time resolution, nodes may receive and send dozens or perhaps hundreds of updates per minute, leading to network and local performance issues at scale. The Market Data Manager module should both place limits on the number of requests it can send and receive per unit time. It should also use functionality from Connection Manager, particularly Peer Discovery, to limit the number of peers it communicates with. This will prevent performance issues on the node level but may still impact Bitcoin's overall network performance as peers are more sparsely connected to each other.

Testing

Both submodules, Rates and Market Data Manager, must be thoroughly tested to prevent them from negatively impacting Bitcoin Core's overall functional and nonfunctional requirements. Unit testing Rates and Market Data Manager during development, as well as integration testing while those submodules are being implemented, will be sufficient to mitigate any significant bugs or vulnerabilities.

Unit testing generally refers to the practice of verifying the functionality of an individual submodule before it is integrated into the rest of the system. In the case of the Rates subcomponent, unit testing should involve thoroughly validating the integrity of its read/write operations in a variety of situations, such as simultaneous read/write requests, missing or malformed data, etc. For Market Data Manager, unit testing should involve verifying that incoming market data is securely parsed and transmitted. This will allow Bitcoin Core developers to quickly identify serious errors before these components are integrated into the rest of the system.

Integration testing must be conducted to verify the functionality of Rates and Market Data Manager in the context of the wider system. Since Rates is connected to Connection Manager in a pub-sub manner, it is unlikely that it will negatively impact its pre-existing functionality. Instead, integration testing should focus on the interactions between Rates and its parent component Storage Engine. For Market Data Manager, integration testing should be performed on the interactions between it and Connection Manager, along with its interactions with other Market Data Manager instances in other nodes when communicating exchange rate data.

Use Cases and Diagrams

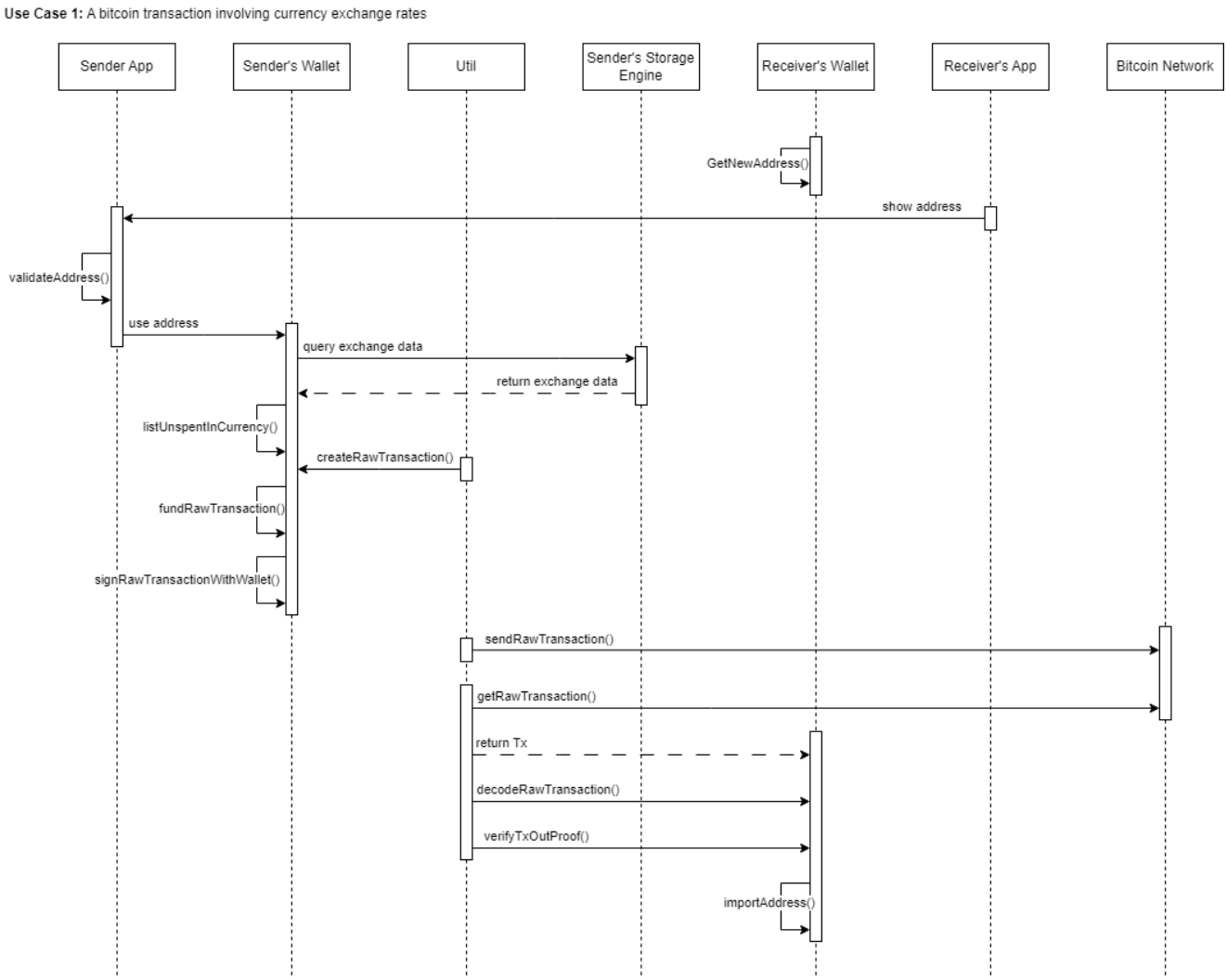

Figure 2. Use case of a bitcoin transaction involving currency exchange rates

The first use case involves a bitcoin transaction between a sender and receiver, except now with currency exchange rates. The transaction starts with the receiver obtaining the address of their wallet and displaying it to the sender through their app. The sender's app reads the address, usually through QR code scanning, validates it, and sends it to their wallet to be used. Before the wallet proceeds, it queries the storage engine to receive the exchange rate data. The storage engine returns this and uses it for the next steps. The wallet gathers its UTXOs (listUnspentInCurrency), and creates a raw transaction funded by the sender's wallet, which then signs it. The user can see how much their transaction will cost them in their preferred currency before it gets sent. The transaction is then sent to the bitcoin network to be validated. The receiver's wallet eventually receives the transaction from the network and decodes it to display human-readable information about the transaction. The receiver's wallet then verifies the transaction output and adds it to their wallet (importAddress), allowing them to spend funds from the transaction output in the future.

Use Case 2: A node updating its exchange rate from a peer

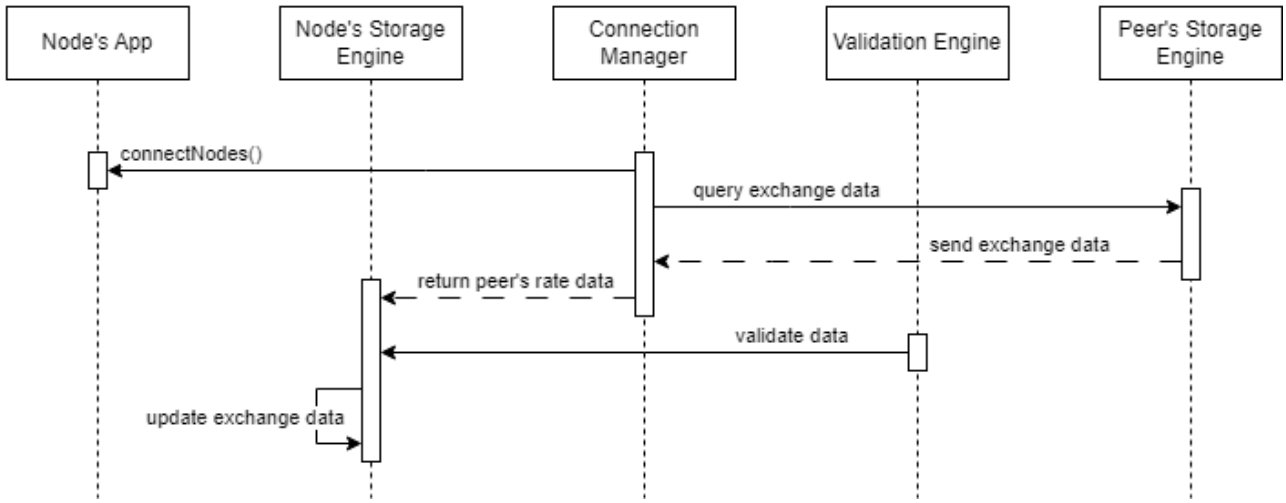

Figure 3. Use case of a node updating its exchange rate from a peer

The second case demonstrates a common functionality of the new addition, which is how nodes can update their exchange rate. The node begins this process from the App component, calling the Connection Manager component to contact other peers. Next, the Connection Manager sends a request to retrieve the exchange rate data from the connected peer's Storage Engine, communicating it to the original node's Storage Engine through the Connection Manager. This data runs through checks from the Validation Engine to ensure the data is safe and accurate. Finally, the node's Storage Engine finishes by updating their exchange data according to the received data. This process is quite simple, which is necessary because in a transaction, sender and receivers must be able to get their exchange rate data quickly to complete the rest of the transaction.

Conclusion

In summary, we present an enhancement to Bitcoin Core in the addition of a submodule of Storage Manager called Rates. We view the addition of this module to each peer as more beneficial than the alternative of implementing ExchangeAPI. Our team conducted SEI SAAM analysis on both implementations, that is examining the salient stakeholders, weighing the respective non-functional requirements, and the impact of the addition on specific subsystems and the architecture as a whole. We identified that the implementation of the Rates module satisfied security, evolvability and usability, three key non-functional requirements for the stakeholders, to a better degree than the ExchangeAPI approach. Following the concreate architecture, the components which would be the most affected by the new feature are the Connection Manager, Storage Engine, Wallet and App. Although, the new feature does not have much impact in general. We advise that unit and integration testing should be employed to mitigate issues, particularly for security. Finally, we showcase two use cases of the system involving transacting and updating with exchange rates implemented.

Lessons Learned

Though researching and proposing new features for a complex software system like Bitcoin Core is a challenging task, it can also provide valuable lessons for software architecture. One of the main lessons the team learned was the importance of modularity. With such a complex system containing many interdependent sub-systems, it's important to design sub-systems in a modular and loosely coupled way. This allows features to be added and maintained as easily as possible. They also learned that when introducing a new feature, it's important to consider its broader implications such as its impact on scalability, security, and decentralization. Through additional analysis and trial and error testing, they ensured that the new feature does not have unintended consequences for the system as a whole. However, architectural lessons are not the only things worth noting.

On top of the architectural lessons, the team also further solidified their understanding of teamwork. For instance, when discussing the implementation of the new feature, various members had drastic ideas from one another. Thus, it was important to come to an agreement about how the team intends to execute it. This helped gain a detailed understanding of the overall solution and kept the team all on the same page moving forward. Lastly, the continued communication allowed them to come up with reasonable deadlines for the group to accomplish. Creating these goals not only allowed them to track their progress, but also ensure that they accomplished everything they set out to do.

Data Dictionary

Transaction – A structure representing a transfer of value from one Bitcoin address to another

Coinbase Transactions – The first transaction of every new block will contain this. It holds the reward for the miner should they solve the block. The reward total is calculated by combining the transaction fees of all transactions in the block together with the reward of the current block height. The block height reward starts off at 50 bitcoin and is halved every 210,000 blocks. Currently, it is at 6.25 bitcoin.

Block – A structure that contains around 1900 transactions and the information necessary for miners

UTXO – An output of a Bitcoin transaction that represents a discrete amount of Bitcoin credited to an address

Blockchain – New blocks reference a "parent" block and thus act as a chain.

Node – A participant in the Bitcoin P2P network, which could be an instance of Bitcoin Core

Naming Conventions

P2P – Peer-to-peer

TX – Transaction

GUI – Graphical user Interface

CLI – Command-line Interface

UI – User interface

PoW – Proof of work

App – The application component

RPC – Remote procedure call

References

Asplund, Mikael, et al. “In-Store Payments Using Bitcoin.” *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, https://doi.org/10.1109/ntms.2018.8328738.

“Chapter 10: 'Mining and Consensus'.” *Chapter 10: 'Mining and Consensus' · GitBook*, https://cypherpunks-core.github.io/bitcoinbook/ch10.html.

“Transactions.” *Bitcoin*, https://developer.bitcoin.org/devguide/transactions.html.

“What Are Bitcoin Blockchain Nodes? - Bitstamp Learn Center.” *Bitstamp*, https://www.bitstamp.net/learn/crypto-101/what-are-bitcoin-blockchain-nodes/.

“Chapter 6: ‘Transactions'.” *Chapter 6: ‘Transactions' · GitBook*, https://cypherpunks-core.github.io/bitcoinbook/ch06.html.