Proposed Lightning Network Integration for Bitcoin Core

by
Gabriel Lemieux (19gml2@queensu.ca),
Jack Taylor (17jmt5@queensu.ca),
Jack Fallows (19jef2@queensu.ca),
Jiacheng Liu (19jl193@queensu.ca),
Maninderpal Badhan (19msb14@queensu.ca),
Nil Shah (20ns3@queensu.ca)
April. 12, 2023

Abstract:

The purpose of this report is to present our proposed Lightning Network integration into the existing open source software Bitcoin Core. Lightning Network integration would yield cheaper and faster transactions for Bitcoin Core users. We outline two possible approaches to achieve this integration, identify important functional and non-functional requirements, and discuss how they would affect the major stakeholders involved in the Bitcoin Core software. In our previous reports we were able to identify and recover both the conceptual and concrete architecture for Bitcoin Core, using these we can specify how our feature would fit into the current system and what components would be affected.

Introduction:

The Bitcoin Core system has revolutionized digital currencies and exchanging money electronically by creating a decentralized system to exchange money. With cryptocurrency growing more popular every year, more and more transactions are being made on the bitcoin network. This influx in transactions is creating bottlenecks causing the network to be less efficient and slower when dealing with new transactions. This in turn is making transaction fees more expensive. To combat this issue we want to add a new feature to the system, a lightning network feature.

This report will start with an overview of the lightning network and its functional and non-functional requirements. (Add requirements). The non-functional requirements for this new feature are: integration, scalability, security, accuracy, reusability and availability.

This will be followed by the different approaches that could be taken to implement a new proposed feature in the current architecture. (small explanation of approaches). By doing a SAAM analysis we figured out which approaches would be ideal. We evaluated the approaches and their impact on the NFRs as well the impact of the NFRs on the different stakeholders of the new feature. From the SAAM analysis we figured out that Approach 1 was the best way to implement the lightning network because it aligns more closely with both the identified NFRs and the spirit of Bitcoin.

The report will be concluded by a sequence diagram depicting a use case scenario of a user creating a new lightning network between him and another user. Followed by a description of the components that will be affected by our chosen approach and how they will be affected .

Proposed Feature:

Overview:

The new feature that our group would like to implement into the Bitcoin Core system is a Lightning Network. This feature would allow for faster transactions, enable more transactions to be made simultaneously, and lower the cost of fees associated with transactions. Currently the Bitcoin Network can handle around 7 transactions per second (Ledger). This may be an issue in the future for Bitcoin to become a viable unit of currency, since the transactions per second are so limited it might hinder the scalability of the Bitcoin Network (Coinbase).

Our group has chosen the Lightning Network as a new feature because the Lightning Network aims to speed up the Bitcoin Network throughput to allow for a major increase in the transactions per second. The Lightning Network allows users to create a transaction channel between each other off the blockchain (Ledger). This channel allows transactions to be done very quickly and since it has been taken off the blockchain the transactions made within the channel are private and off the public ledger . To create a channel in the lightning network the two users who wish to open a channel between each other need to create a funding transaction on the blockchain. Once the channel is created it allows for extremely efficient communication between parties on the channel (CoinTelegraph). In addition not every pair of users needs to create a new channel, since the network can use intermediate channels to send money instead of initializing a new channel each time (CoinTelegraph).

Functional Requirements:

The main functionality for the lightning network will be to create a peer-to-peer payment channel between parties that is off of the blockchain. Our approach would be to create a lightning network module and in the module there will be subcomponents of: a network submodule to connect different parties in a lightning network, a submodule to create the smart contract that's required on creation of the channel, a transaction submodule to handle the transactions on the lightning network, and a ledger submodule to keep track of the transactions.

For the network submodule, it will be given the addresses of the nodes who want to create a lightning network channel along with an initial amount of BTC locked in and it will output a separate channel. It will have a dependency on the smart contract submodule to create a smart contract on the instantiation of the channel. Also on instantiation of the channel, the

network submodule will communicate with the Bitcoin Network to transmit the opening of the channel to other nodes on the network. Finally, on the closing of the channel the network submodule will send the output of the transactions submodule – which contains the ledger – to the Bitcoin Network so the sum total of the transactions can be transmitted to the Bitcoin ledger as one transaction.

When a lightning network channel is created, the smart contract submodule will be used to take any preset requirements that the users agree on and output a smart contract between the parties to make sure the requirements are fulfilled automatically on the channel (Coinbase).

For the transaction submodule, it will take the input of the two nodes who want the transaction, verify the authenticity of the transaction, and output the transaction to the ledger submodule.

The ledger submodule will simply be a submodule to keep the channel ledger updated, transactions will be inputted into the ledger submodule from the transaction submodule, and the ledger submodule will update the channel's ledger. When the users want to close the channel, the ledger will take the sum total of the ledger to anonymize the transactions, so that the sum total transaction can be outputted as a single transaction on the main Bitcoin Ledger.

Non-Functional Requirements:

Performance - Since the goal of the lightning network is to increase throughput, the lightning network must be able to handle a large amount of transactions per second (CoinTelegraph).

Scalability - This feature will allow for more transactions to be made at the same time as only the opening and closing transactions will be recorded and validated on the blockchain, while other smaller intermediate transactions are performed on the Lightning Network (CoinTelegraph).

Security - Since only the opening and closing transactions will be on the blockchain, the other transactions made in the Lightning Network will not be validated by the blockchain and nodes in the network. This means the risks of being cheated by the other user in the lightning network are higher. The new feature should allow users to close the network whenever they feel a risk.

Accuracy - The opening and closing transactions on the blockchain should depict $\sum$ all money sent to user A - $\sum$ of all money sent to users B in the Lightning Network accurately.

Maintainability The feature should be designed for easy updates and bug fixes, without the need for significant code changes for each new Lightning Network update

Availability - The Lightning Network will need to be available to users 24/7 since people from any time zone will be wanting to make transactions whenever they can and people could have lightning networks open for weeks/months
Integration - Each different lightning network should be well integrated into the bitcoin core system.

Proposed Approach:

Approach 1:

Our main approach to integrating the lightning network would be to create a new lightning network module in the core architecture. This module would have submodules of: a network submodule, a submodule to create the smart contract, a transaction submodule, and a ledger submodule. The lightning network module will have dependencies on the connection manager to use the Bitcoin Network and to make use of networking/http related functionality to connect to the blockchain and the Bitcoin Network. It will have a dependency on the Peer Discovery to communicate with peers to create channels. It will have a dependency on Wallet to access the private/public keys. It will be required on the transactions module for common transaction functionality. It would depend on the RPC module as the network submodule would use the RPC as a direct connection to Bitcoin core. Also, it would depend on the Transaction module to access the common functionality of signing transactions with private/public keys.

Approach 2:

Instead of creating a new submodule, an alternate approach could be to extend the connection manager and implement the lightning network as a submodule of connection manager. We would implement the lightning network functionality inside the connection manager by extending the existing submodules and adding a new submodule to add new functionality. The new submodule would implement the smart contract and update the ledger. The initial fee lock-in could be implemented in the node submodule. Connecting to the block chain and opening/closing the channel would be implemented in the HTTP Server submodule. Outgoing dependencies would be similar to approach 1.

SAAM Analysis:

In Table 1 we discuss how the approaches outlined above compare to our identified Non-Functional Requirements for this new Bitcoin Core Feature. This will aid in our decision of our final approach based on which more closely aligns with our NFRs.

Table 1: Comparing the two approaches align with identified Non-Functional Requirements

NFR	**Approach 1**	**Approach 2**

Integration	By creating a new module you may simplify the integration of Lightning Network features into Bitcoin Core, however introducing its dependencies it needs functionality from may introduce challenges	Integration of Lightning Network functionality as a submodule of Connection Manager may prove challenging as Connection Manager is a central feature of Bitcoin Core with many moving pieces
Security	By isolating Lightning Network functionality into its own component you minimize the risk of security vulnerabilities in the component spreading to other components of Bitcoin Core due to a stronger decoupling of code.	A submodule approach may introduce more security risks due to tighter coupling between components
Scalability	Creating a new module may make it easier to isolate Lightning Network functionality which could allow for easier scaling without touching other features, but on the other hand a new module introduces more complexity than a submodule which may impact the systems overall scalability	Making the Lightning Network a submodule within Connection Manager would lead to a tighter coupling of the two functionalities which may complicate both the scaling of Connection Manager and Lightning Network features.
Availability	By using a separate module approach you may reduce the availability of the Lightning Network as it has a heavy dependency on the Connection Manager in order to facilitate transactions. The increased dependencies it requires may introduce an additional point of failure which would decrease the availability.	A submodule approach may introduce availability issues with the Connection Manager module if the Lightning Network were to go down which would affect the continuous function of Bitcoin Core. On the other hand the reduced number of dependencies and easier access to incoming information may minimize the potential points of failure
Accuracy	If the Lightning Network were to be decoupled from the Connection Manager there is a chance that it wouldn't have the most accurate and up to date information as it relies on intercommunication between components.	Close coupling of the Connection Manager with the Lightning Network would likely lead to more accurate real time data on transactions and user requests.
Maintainability	A new module would likely be	A sub module may be more difficult

	easier to maintain due to isolation from other components.	to maintain due to coupling, affecting both Connection Manager and Lightning Network functionality when changes are introduced to either.

In Table 2 we examine the relationships between each of our stakeholders and the identified NFRs. By performing this analysis we hope to gain insight into the most important NFRs for each stakeholder so we can make a more informed decision on our final approach.

Table 2: Comparing the relevance of different Non-Functional requirements to stakeholders

NFR	**Developers**	**Users**	**Merchants**	**Miners**
Integration	The lightning network will be easy to integrate into the blockchain component since the opening and closing transactions are recorded onto the blockchain	Should be integrated in a user friendly way, allowing for a cohesive feature set/interface	The new feature will be integrated into the bitcoin core system which will allow any merchant to create a lightning network and there will be no need for specific wallets	Integration of opening and closing transactions into the pre-existing block validation process without complications is essential for Miners
Security	Few security vulnerabilities introduced that were not already present in the previous implementation	The feature will allow any user to close the lightning network between them and another user if one is uncooperative	There should be a secure handling of the Bitcoin payments without any risk of fraudulent transactions	Protection against possible fraudulent transaction attempts like double spending
Scalability	The ability to support an ever increasing number of new Lightning Network transactions/channels	The feature will allow users to create more transactions at the same time.	They need to ensure they can match customer demand during peak transaction periods, processing a high volume of transactions without bottlenecks or delays	There should be a minimal impact on the overall size of the blockchain, as most Lightning Network transactions occur off chain, there is only the need for opening an closing transactions

Availability	Ensure that the Lightning Network remains functional within Bitcoin Core even when it undergoes software updates or bug fixes	Users will be able to create transactions through the lightning network can be 24/7 and open/close the whenever they want	The Lightning Network must be continuously available to process customer payments without any interruptions. This is very important for customer satisfaction and profit	There must be a continuous availability of processing Lightning Network channel opening and closing transactions
Accuracy	Since only the opening and closing transactions are recorded on the blockchain. The values displayed on the blockchain will need to accurately depict the transactions inside the lightning network	The lightning network will be 100% accurate and once the lightning network is closed the transaction that goes onto the blockchain will be correct and will contain the right amount	There should be no lost profits due to incorrect transaction fee calculations or errors in opening or closing transactions.	Precise validation of channel opening and closing transactions on the blockchain, the distributed ledger
Reusability	The lightning network component will not require new code to be added for every new lightning network create and will reuse the blockchain component to add the opening and closing transactions onto the blockchain	The user should not need to make changes to their wallets to allow the use of lightning networks	Merchants should not have to update their transaction implementation every time Lightning Network is updated	Miners should not be affected by lightning network opening and closing transactions appearing on the blockchain

Approach 1 seems to favor Integration as creating a new module is easier than locating all the places in a pre-existing module where you need to add or extend code to fit your needs, a new module could be built from scratch to have the best possible structure for the new features

functional requirements. However if Connection Manager already has ties to other modules, it might make it easier to modify the existing code to accommodate the Lightning Network. Approach 1 also favors security, as it separates the new features code from the rest of the code. Tightly coupling the Lightning Network with the Connection Manager as Approach 2 requires may lead to security vulnerabilities in a central component to Bitcoin Core. Approach 1 also favors scalability due to decreased complexity of possible interconnected bottlenecks. Having code that intermingles functionality, as in Approach 2, may mean changing code to address a scalability issue in the Lightning Network in one place could cause a bottleneck in the Connection Manager in another, introducing a web of complexity. Approach 2 seems to favor Availability and Accuracy more than Approach 1, having the Lightning Network closely linked to the connection manager reduces the total dependencies between modules which minimizes the points of failure and provides more up to date information. Finally Approach 1 favors Reusability/Maintainability over Approach 2 as a new module would have much clearer bounds that something closely coupled to an already complex module.

Based on our analysis we believe Approach 1 is the much stronger choice, it adheres more closely to both the NFRs and the spirit of Bitcoin. Having an off chain transaction feature be easily disabled instead of closely coupled may be preferred by users who only want public on chain transactions. Some users also view Bitcoin as being the most pure representation of layer 1 blockchain technology and may be opposed to close integration with a layer 2 technology.

Changes in Architecture/Affected Components:

Our main approach is creating a new module that would depend on the other modules. Thus the main changes in architecture would be updating dependencies and implementing a new module.

Connection Manager

The connection manager would have a bidirectional dependency with the lightning network module. We have to add functionality to the connection manager to allow each channel to connect with the Bitcoin Network when each channel opens/closes. The connection manager would depend on the lightning network, because when other peers need to verify the lightning network transaction on the channel close, they would need to access the lightning network module to access lightning network functionality.

Transactions

There would be a one-way dependency from the lightning network module to the transaction module. No changes would be made directly to the transaction module, but the lightning network would require functionality from the transaction module. When a transaction is put on the lightning network channel, it must be signed with the private/public keys. The

transaction module already contains functions to do this, thus the lightning network module would reuse these functions.

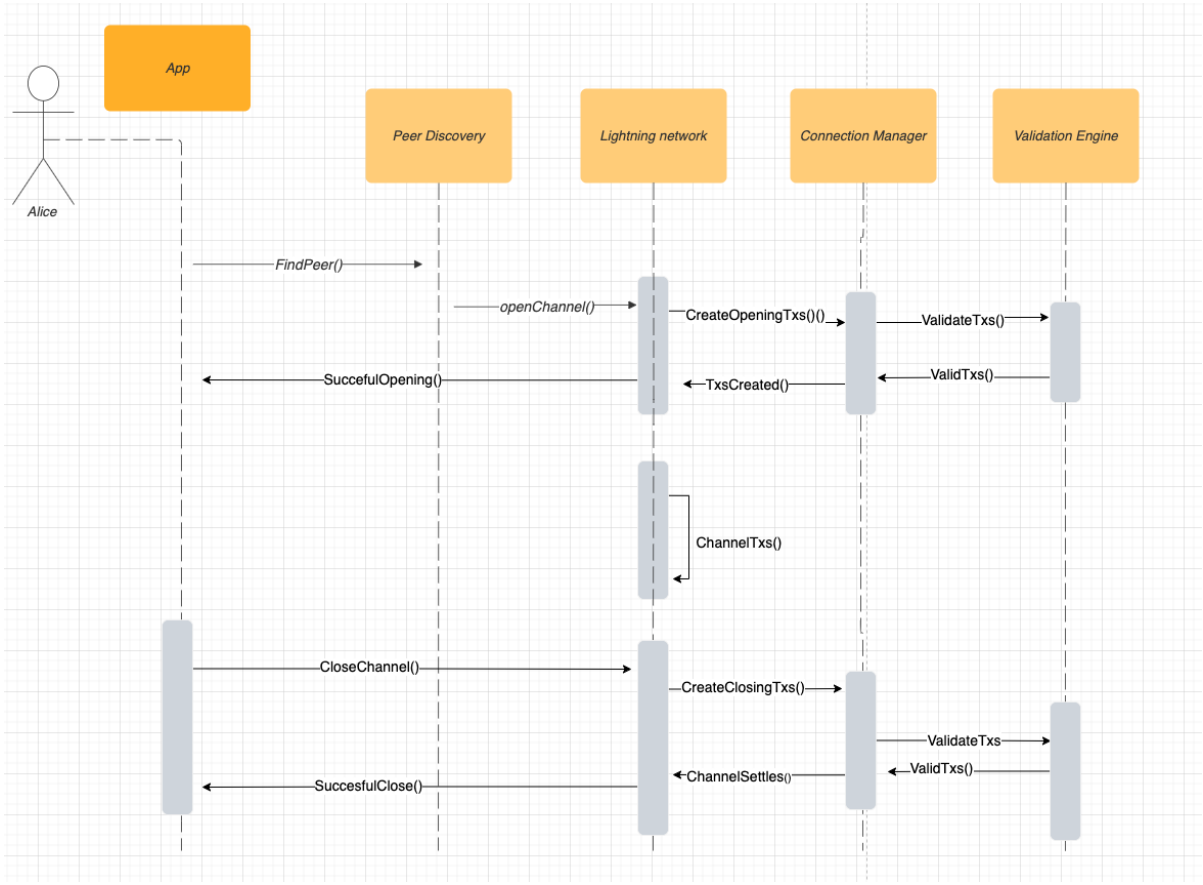
Wallet

There would be a one-way dependency from the lightning network module to the wallet module, since the lightning network module needs to access the private/public keys to sign the transactions on the lightning network ledger. The wallet would have new functionality added to create a new private/public key for every lightning network channel.

Peer Discovery

There is a one-way dependency from the lightning network module to the peer discovery module. There would be no added functionality to the peer discovery module, but the lightning network would use the peer discovery module to access known peers to easily create new lightning network channels with them when needed.

Use Cases:



This sequence diagram illustrates the use case of a user named Alice creating a channel in the lightning network with another user, making transactions inside the channel with that user and then finally closing the channel.

Testing:

For testing the newly created dependencies between the submodules, we could create unit tests where we create mock lightning network channels between mock peers and have mock transactions between these peers. We would confirm first that all the functional requirements are met. In the unit tests we will check first that the lightning network module can open and close a channel on the Bitcoin Network. We will make sure that peers are able to connect to each via peer-2-peer connections and are able to create a new private lightning network channel between each other. We will ensure that the submodule is able to successfully create a smart contract on creation as well. We will verify that the peers are able to send transactions on the lightning network and are able to successfully sign with their private/public keys.

Potential risks:

The main risk is that the initial funds sent to the lighting network channel can get stuck due to technical issues (CoinTelegraph). A lot of unit testing has to be done before the implementation of the lighting network to make sure that this doesn't occur. Both parties in a lightning channel transaction also need a sense of morality and not close the channel without informing the other party (CoinTelegraph).

Data Dictionary:

Block - A block is a collection of bitcoin transactions that have been validated and recorded on the blockchain. Each block contains the transactions, a header with other metadata, and a reference to the previous block in the chain.
Block Header - Contains information about a given block and its transactions. Block Headers are hashed repeatedly during proof of work.
Blockchain - the decentralized, immutable, public ledger of all bitcoin transactions, stored in cryptographically linked blocks of transactions.
Decentralized applications - application that runs on a decentralized network instead of a single server
Genesis Block - The first block in the Bitcoin blockchain.
Merkle Tree - also known as binary hash trees, a Merkle Tree is a tree that serves as a summary of all the transactions in a block by providing hashes for different blocks of data at its leaf nodes.
Minting - Minting is the process of creating more Bitcoin during the addition of each new block.
Mining - Validation of pending transactions within the Bitcoin network.
Miner Pools - A network of distributed miners who split rewards for a successful block solution based on computational resources provided.
Nonce - a value that is set so the hash of a block will have a number of leading zeros

Proof of Work - A form of cryptographic proof used to show that a given amount of computational effort has been used to reach a solution. The proof is easily verifiable by independent parties once it is presented. In the case of Bitcoin, Proof of Work means that a Miner has presented valid parameters to SHA256 that solve the current Block.
Full Node - A program that fully validates transactions and blocks. Almost all full nodes also support the network by accepting transactions and blocks from other full nodes, validating those transactions and blocks, and then relaying them to further full nodes.
DNS seed - A DNS server which returns IP addresses of full nodes on the Bitcoin network to assist in peer discovery.
SHA256 - The hash function used in Bitcoin Core's mining process. SHA256 always produces a 256-bit output regardless of input size. This can be thought of as a way to create unique fingerprints for every input.
UniValue: Abstract Data Type which can represent a null, string, bool, int, array container or key/value dictionary which uses JSON encoding and decoding

Naming Conventions:

UTXO - Unspent transaction output
STXO - Spent transaction output
TXs - Transactions
P2P - Peer-to-peer
NFR - Non functional requirement
FR - Functional requirement
QA - Quality Attribute

Lessons Learned:

Through the process of identifying both functional and non-functional requirements of a new feature we gained an appreciation for the difficulty and nuance in introducing a new feature into such a large system.

Since Bitcoin Core is an open source system, when adding a new feature it will be very important that everyone is clear about how it will be integrated in the system and all the requirements. If some information is left vague and unclear it could cause major issues in the development of the new feature which will make

When the new feature has various stakeholders it can be difficult to find an implementation that will please everyone's non-functional requirements and you have to decide which are more important for the feature to follow.

Conclusions:

This report has presented a comprehensive proposal for integrating the Lightning Network into Bitcoin Core to address the growing challenges of transaction speed, capacity, and cost. We have identified and summarized the functional and non-functional requirements for the Lightning Network module, which include performance, scalability, security, accuracy, reusability, maintainability, availability, and seamless integration with the existing Bitcoin Core system.

After evaluating two potential approaches using SAAM analysis, Approach 1 was recommended as it better aligns with the identified requirements and the spirit of Bitcoin. This approach involves creating a new Lightning Network module with dependencies on Connection Manager, Peer Discovery, Wallet, Transactions, and RPC modules, with necessary architectural changes outlined for each of these components.

In summary, the proposed Lightning Network integration will significantly enhance the capabilities of Bitcoin Core, enabling faster and cheaper transactions while maintaining the decentralized nature of the platform. As future directions, we recommend conducting thorough testing and validation of the proposed architecture, as well as soliciting user feedback to further refine the Lightning Network module. The successful implementation of this feature will pave the way for a more scalable, secure, and efficient Bitcoin ecosystem, ultimately benefiting all stakeholders involved.

References:

Bitcoin Core architecture. (n.d.). Bitcoin Core Architecture. https://jameso.be/dev++2018/#1

Index. (n.d.). Bitcoin Core. https://bitcoincore.org/en/doc/24.0.0/

Bitcoin Core: Main Page. (n.d.). Bitcoin Core: Main Page. https://doxygen.bitcoincore.org/index.html

B. (2023, March 24). GitHub - bitcoin/bitcoin: Bitcoin Core integration/staging tree. GitHub. https://github.com/bitcoin/bitcoin

Chapter 3: 'bitcoin core: The reference implementation'. Chapter 3: 'Bitcoin Core: The Reference Implementation' · GitBook. (n.d.). Retrieved March 24, 2023, from https://cypherpunks-core.github.io/bitcoinbook/ch03.html

Lightning Network. (n.d.). Retrieved April 12, 2023, from https://lightning.network/

Coinbase. (n.d.). *What is the Lightning Network?* Coinbase. Retrieved April 12, 2023, from https://www.coinbase.com/learn/crypto-basics/what-is-lightning

Cointelegraph. (2023, March 27). *What is the Bitcoin Lightning Network, and how does it work?* Cointelegraph. Retrieved April 12, 2023, from https://cointelegraph.com/learn/what-is-the-lightning-network-in-bitcoin-and-how-does-it-work

Transactions per second (TPS). Ledger. (2022, December 16). Retrieved April 12, 2023, from https://www.ledger.com/academy/glossary/transactions-per-second-tps#