Exploring Off-Chain Transactions in Spartan-Gold

Valerian Salh
CS168-01 Blockchain and Cryptocurrencies
San Jose State University
Dr. Thomas Austin
6 December 2021

I. Introduction

Bitcoin is somewhat notorious for its strict limits on the size of blocks and transaction volume. Transaction volume on the network is limited to just 7 transactions per second, while the size of a block is limited to 1MB [1]. For comparison, during the holiday season of 2013, Visa's transactions per second peaked at 47,000. For Bitcoin to manage a similarly equivalent volume, each block would need to store eight gigabytes worth of transactions per block [3]. This is a subset of the issue that many argue to be one of Bitcoin's biggest caveats: the scalability problem.

The limits on transaction volume and block size are not the only issues concerning the scalability problem. Transaction fees pose another threat to the success of the network. If Bitcoin were to be widely adopted for transactional use, transaction fees would skyrocket. Due to the nature of the blockchain and its use of a public ledger, every transaction on the network must be validated and added to a block. With the network's limits on volume, transactions would need to compete for recognition using fees [1], thereby eliminating the practicality of using the network at all.

Some altcoins have implemented changes to Bitcoin's limits by forking the main network, such as Litecoin, while others have opted to operate completely differently. However, some proposed solutions to the scalability problem do not involve forking or

creating another network altogether, but rather operating on top of Bitcoin's existing network. The Lightning Network for Bitcoin is currently a popular implementation to function in this way and it also happens to be the inspiration for this project. Solutions such as the Lightning Network are known as "second-layer solutions," in reference to Bitcoin being the "base-layer."

II. The Bitcoin Lightning Network

There are two main aspects to the Lightning Network for Bitcoin. The first is functionality for off-chain transactions via a bidirectional payment channel. The second is the network itself, which comprises many different bidirectional payment channels [4].

A bidirectional payment channel works by being created off-chain. First, two parties enter an agreement to create a channel. A 2-of-2 multi-signature address is created from the information of the two parties (such as their public keys) [5, 6]. One or both parties transfer a select amount of Bitcoin to the multisig address. This transaction takes place on the Bitcoin blockchain. At this stage, both parties have locked away funds inside the multisig address and these funds are no longer accessible on-chain. The bidirectional payment channel now holds their respective funds that were transferred to the multisig address. On this channel, the two parties can transfer and receive funds a seemingly limitless number of times. None of these transactions are recorded on-chain, and therefore none of these transactions contribute to Bitcoin's transaction volume [2]. However, only funds that have been locked away in the channel may be used and no new funds may be added. As a result, it is possible for a party in the channel to exhaust their funds. Every transaction made in the channel is signed and agreed upon by both parties. Finally, when the channel is closed, the latest balances within the channel are posted to

the Bitcoin blockchain via another transaction, where the multisig address pays the two parties involved in the channel. It should be noted that only one party is needed to close the channel, and when the channel is closed, the most recent valid balance sheet is what is published to the Bitcoin network [4]. Alas, with the channel closed and the involved parties having been paid out by the multisig address, a vast history of transactions have been reduced to just two.

The Lightning Network is composed of the bidirectional payment channels described earlier. Because it is a network, a path between channels can be found and used to allow two parties to transact with each other without having a direct payment channel between them. The Lightning Network will find an optimal series of intermediary channels that possess adequate funds to create a transaction between a source and endpoint. No node along the path is trusted and the transactions are enforced with scripts that use decrementing time-locks to enforce atomicity [4]. The Bitcoin blockchain serves as the arbiter for these transactions because all transactions can be published to the blockchain, where they are subject to Bitcoin's scrutiny.

The approach of the Lightning Network significantly reduces transaction fees and costs, with transactions made through the Lightning Network being incredibly cheap (at the time of writing this). Thus, the issue of scalability is addressed, as well as the issue of fee pricing.

III. Limitations and Scope

My project is known as Hoplite. For the scope of this project, I was tasked with implementing off-chain transactions and simulating something akin to the Lightning Network. I successfully implemented off-chain transactions, but did not simulate the

notable functionality of the Lightning Network due to time constraints and other commitments. The off-chain transactions are implemented similarly to the bidirectional payment channel described in section II.

It should be noted that certain liberties and leniencies were taken in regards to security since this is a simulation. Primarily, the payment channel that is created has access to the keypairs of the clients involved and the private keys of the clients are returned together after a channel is created. Naturally, this poses a potential risk in that the private key of a party could be compromised if the other party (or any malicious party) obtains it. Private keys are needed to make transactions on the payment channel, thereby giving power to the malicious party to make as many transactions as they'd like on behalf of the compromised party.

IV. Implementation

My implementation of the bidirectional payment channel is limited in comparison to the specifications of the Lightning Network. For starters, I did not make use of a multisignature address, instead opting to simply create a new address based on the methods already supplied by the UtxoClient class of Spartan-Gold.

One of the reasons for a simple approach is because the complexity of properly creating a functional 2-of-2 multisignature address made from the public keys of two parties felt unnecessary for the scope of this project. Because the payment channel inherits from the UtxoClient class, I am able to create a wallet with a single address that will act as the locked funds pool for a payment channel. Inheriting from the UtxoClient also made posting transactions to the blockchain significantly easier, since I did not need to create a new system to properly publish verifiable transactions to the network.

My earlier approaches to the task involved creating many classes, where I even considered creating a more advanced system of transacting that involved a history of two-party-signed transactions (much like a ledger), as well as a completely distinct client class designed to be used only on the payment channel that did not inherit from any of the other client/miner/mixin classes. I even explored making changes to the base UTXO model of Spartan-Gold. However, these approaches felt too time-consuming, overly difficult, and seemed counter-productive. I opted to make virtually no changes to the UTXO model of Spartan-Gold, and instead made just one PaymentChannel class to handle transactions off-chain. I avoided modifying Spartan-Gold to better fit the theme of emulating the Lightning Network. Much like the Lightning Network is a second-layer solution that operates on top of the Bitcoin network, my project operates on top of UTXO Spartan-Gold.

The payment channel works by first being constructed as a UTXOClient and then being registered on the blockchain and fake net. A channel can then be created by passing the two clients that will be in the payment channel, as well as the amount of gold they would like to commit to the channel. The channel will then fund its address by the amount specified by the clients, and will update its internal balance sheet to reflect their initial deposits. Creating the channel returns the address and private key associated with the address that will be paid after the channel closes.

When transactions are made on the channel, the channel client object is called to make a transaction, specifying who the payment is from and the amount being transferred. It does not need to be specified who the payment is to because these channels are strictly for two parties. The channel will create the transaction, and then use a

callback function to get a signature from the client who made the transaction. The transaction is then validated and verified by the channel class, where if successful, will then be used to update the balance sheet of the channel.

When the channel is closed, the channel will post a transaction to the network reflecting the most recent valid transaction, and the channel will pay the two involved parties their respective amounts. The transaction fee is deducted from both parties equally. A small modification was made to the blockchain class of Spartan-Gold, where I changed the default transaction fee to be 2 instead of 1, just for the sake of keeping whole numbers.

In conclusion, I successfully implemented a functional bidirectional payment channel off-chain that allows the two parties in the channel to make transactions with each other numerous times, very quickly and without fees.

V. Results

The payment channel I implemented proved to be very successful in comparison to transactions that operate on the network. Transactions were significantly faster when done through the payment channel and were free; in fact, I ran into issues dealing with Javascript's asynchronous nature because the Spartan-Gold network could not process transactions fast enough for the payment channel to work initially. After payment channels are created, clients can transact with each other limitlessly at a much faster speed. Furthermore, clients may participate in more than one channel at a time. So long as the channels are successfully created, the main blockchain should reflect that they no longer have access to funds that have been locked away. From there, clients may make

transactions on either of the payment channels back to back with no issues. I did have to pay special attention to using timeouts to allow time for creation of payment channels and the transfer of funds from clients to the payment channels.

I failed to emulate the full functionality of the Lightning Network due to time, but I did initially explore how I might start with coding it. HopliteNetwork.js details my brainstorming on implementing the channel-to-channel routing functionality of the Lightning Network. I may revisit this project in the future to better flesh out its functionality.

Overall, I am relatively satisfied with what came of the program in the time I spent on it. However, I am disappointed in my final approach and believe that there are improvements to be made, starting with my over-reliance on the driver (channel-test.js) to initialize fields and support transactions. Ideally, I would want to create a special client class that will be granted more control over their participation in the Hoplite Network. This new client class would have it's own methods to create transactions and prompt other clients to also sign off on them. Furthermore, it would be nice if this new client class had control over closing their payment channels. Naturally, a fully functioning Hoplite Network would be the biggest improvement to be made. Irregardless of my disappointments, the project was still insightful and enjoyable.

References

[1] SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. https://jbonneau.com/doc/BMCNKF15-IEEESP-bitcoin.pdf

[2] What is the Lightning Network? https://river.com/learn/what-is-the-lightning-network/

[3] What is the Lightning Network in Bitcoin and how does it work? https://cointelegraph.com/bitcoin-for-beginners/what-is-the-lightning-network-in-bitcoin-and-how-does-it-work

[4] Lightning Network. https://lightning.network/

[5] Understand how Bitcoin multi signature work by Javascript. https://medium.com/@t.tak/understand-how-bitcoin-multi-signature-work-by-javascript-18ed10f35941

[6] Multi-signature. https://en.bitcoin.it/wiki/Multi-signature