Probabilistic Lightning

Jamie BLOXHAM [a], Gina YUAN [a], Andrew XIA [a], Justine JANG [a]
[a] *Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*

Abstract. With regular Bitcoin transactions, low-value, high-frequency payments are increasingly impractical due to increasingly significant mining fees that must be paid with each transaction. The Bitcoin Lightning Network is an extension to Bitcoin that allows two parties to create a payment channel between themselves, allowing payments to be made without committing many transactions to the blockchain, thus avoiding substantial mining fees. However, these payments still cannot be smaller than a satoshi, the smallest unit of Bitcoin. In this paper, we describe a scheme for probabilistic payments in the Lightning Network, which can be utilized to effectively make sub-satoshi microtransactions.

Keywords. Bitcoin, Lightning Network, microtransactions

1. Introduction

The Bitcoin Lightning Network is a research project currently ongoing at the Digital Currency Initiative as part of the MIT Media Lab. The Lightning network[1] is a set of nodes linked by 2-party payment channels built from Bitcoin smart contracts. The software is currently being built out and initially tested on the Bitcoin Testnet[2].

For this project, we will focus on implementing probabilistic payment systems on the Bitcoin Lightning Network. Beyond enabling trivial gambling-esque systems, probabilistic payments more notably present an opportunity to emulate payments of amounts below one satoshi, the minimum for Bitcoin. We also looked into other vulnerabilities like flaws in the user interface, codebase implementation, and theory that compromised the security of the system.

2. Bitcoin [3]

Bitcoin is a decentralized digital currency system that can be used to send payments to anyone around the world. The supply of bitcoins is regulated only by software and the agreement of users. Bitcoin uses blockchain technology, a decentralized public ledger, to make all past transactions publicly available. Transactions are committed to the blockchain when a new block is mined that contains the transaction. They can also be verified by investigating the spent coins' path through the entire block chain.

Given that blocks on the blockchain are only added and never removed, it is inevitable that Bitcoin is going to run into scalability issues. The size of the blockchain has grown exponentially since it first started, and is now over 100 GB (Figure 1). To

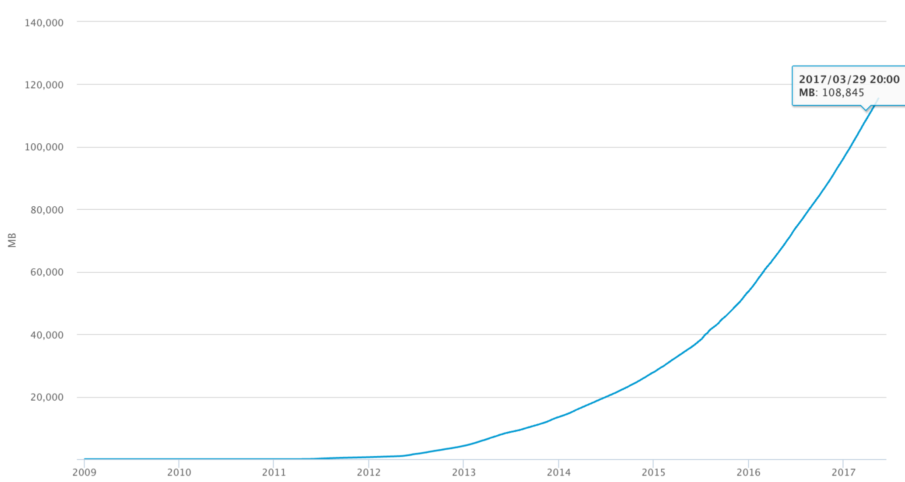

Figure 1. The size of the Bitcoin blockchain since its origin in 2009. Blockchain size has seen exponential growth over the past few years, and has now reached over 100GB. (Graph from blockchain.info)

prevent transaction spam, miners now generally expect a small fee to be included with each transaction. Even with preventive measures like this, Bitcoin is rapidly becoming unviable for low-value high-frequency transactions.

3. Bitcoin Lightning

Bitcoin Lightning was proposed in 2016 by Poon and Dryja as a method to solve the scalability problem of sending microtransactions between two parties [1] [2]. In a *Lightning payment channel*, two parties —Alice and Bob —may transact amongst each other off the chain. If Alice and Bob were using the main blockchain to broadcast and record their n transactions, there would be mining fees for all n transactions, and the cost would accrue significantly. However, by using the Bitcoin Lightning network, a constant number of broadcasts, the initial channel funding transaction and the final commitment transactions that would close the transaction, would be broadcasted to the main blockchain. In this sense, the Lightning Network can both help reduce "spam" on the main blockchain and also reduce the overhead costs for transacting parties.

For the purposes of explanation, let Alice and Bob be two parties that wish to create a lightning channel between the two and transact. All transactions that will be broadcasted onto the main blockchain must be signed by both Alice and Bob. We define the *state* of a payment channel as a distribution of money agreed upon by both parties. For example, if Alice has 0.5BTC and Bob has 0.5BTC at state 1, and Bob decides to send Alice 0.1BTC, then in state 2 Alice would have 0.6BTC and Bob would have 0.4BTC.

There are two main types of transactions. A *funding transaction* is broadcasted onto the main blockchain, signed by both Alice and Bob in a 2-of-2 multisignature script, such

State	Alice		Bob	
1	0.5	C1a	0.5	C1b
2	0.6	C2a	0.4	C2b

Figure 2. An example channel between two parties, Alice and Bob, reflecting the state of their channel

that the main blockchain can acknowledge that a new payment channel holding a certain amount of money, has been created.

A *commitment transaction* is a transaction that allows either party to unilaterally close a channel at a certain state of the channel. Every commitment transaction spends the funds created by the funding transaction, though only one is broadcasted to the blockchain. For each state, Alice and Bob have distinct commitment transactions. Because all transactions are signed by both parties, each party must obtain the other's signature for their own commitment transaction. That is, Alice needs Bob's signature for her commitment transaction for each state. A party can broadcast their commitment transaction to close the channel, thus depleting the funds allocated by the funding transaction, and causing the broadcast of other commitment transactions to be registered as double spends. However, if a channel is incorrectly or malevolently closed at an obsolete state, then the party broadcasting the incorrect state will be penalized and lose all of their money. We will explain the details of the commitment transaction and how to cash out the money in the channel in section 3.2.

3.1. Creating a Lightning Channel

To create a Lightning payment channel, Alice and Bob will initially decide how much money to put in the channel. Let us say that Alice and Bob both put in 0.5BTC in their shared channel, such that the channel is worth 1BTC in total. First, Alice and Bob must first exchange their signatures for commitment transactions for state 1, where Alice and Bob both have 0.5BTC. Only after they have exchanged these signatures can the two broadcast the funding transaction, stating that the channel has been created. This is because if Alice and Bob are uncooperative, then if they broadcast their shared channel onto the main blockchain, and then decide not to exchange signatures for each others' commitment transactions, then neither party will be able to reacquire their money in the channel.

Figure 5 describes an example channel and the associated states between Alice and Bob. Note that at each state, Alice and Bob both have commitment transactions, signed by the other party, that either can unilaterally broadcast to close the channel.

3.2. Transacting in a Lightning Channel

Now that the Lightning payment channel has been created, Alice and Bob wish to send funds between each other on the channel. Transactions between Alice and Bob will increment the state of the channel without having to broadcast anything onto the main

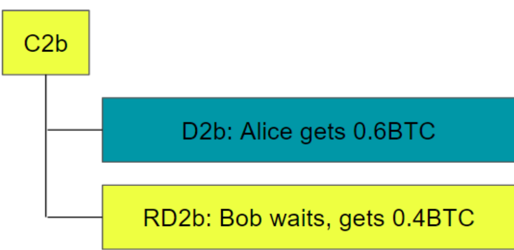

Figure 3. A valid channel closing, referring to the state of the channel as described in figure 5

blockchain. In figure 5, if Bob decides to send Alice 0.1BTC following the initial funding state, then at state 2 Alice will have 0.6BTC and Bob will have 0.4BTC.

When a commitment transaction has been broadcasted onto the main blockchain, the channel has closed. Each transactions splits the funding input into two outputs, one for each party. These outputs are asymmetric in function, although under normal circumstances both parties are able to claim their respective outputs. Let us assume, for sake of demonstration, that Bob is broadcasting his commitment transaction C2b. One output is given to a ”pay to pubkey hash” (P2PKH) payment address corresponding to Alice; essentially this is a regular Bitcoin transaction output which Alice only needs to sign. This is called the ”deliverable”, denoted D2b for the commitment transaction C2b. The other output is a ”pay to script hash” (P2SH) payment address. Bitcoin P2SH payments require the receiving party to provide inputs that allow a provided script to evaluate to true. For this output, the script is an OR that can be satisfied by one of two conditions. The first of these conditions is simply for Bob to sign after some amount of time has passed since the broadcast of the transaction. This is enforced using the ”OP_CHECKSEQUENCEVERIFY” opcode available in Bitcoin scripts. This output, when claimed in this way, is called the ”revocable delivery”, denoted RD2b in this case.

It may, however, be advantageous (and dishonest) for one party to broadcast not the latest state in the channel (for example, in figure 5, Bob is inclined to broadcast commitment transaction 1). The Lightning Network combats this by allowing a party to claim all money placed in a payment channel if they notice that the other party has attempted to cheat them by broadcasting an old commitment transaction. Say, for instance, that Bob broadcasts C1b, after both parties have agreed upon state 2. Recall that Bob must wait a period of time before he is able to claim his output from this transaction, and that this was only one way that this output could be claimed. The other condition in the script enables Alice to claim it! This is known as the ”breach rememdy”, denoted ”BR1b” in this case. To prevent Alice from utilizing the breach remedy for a current, and thus valid, state, Alice is required to sign with a secret generated by Bob. This secret is unique to the state, and both parties exchange their old secrets when they advance to a new state, thus ”revoking” their claims upon the old state.

In figure 3, we see an example of a valid commitment transaction behind broadcasted, and the following events that would happen afterwards to allow Alice and Bob to acquire their correct funds. Because state 2 is the most up to date state, after Bob broadcasts his commitment transaction, Alice, can immediately sign her delivery output to attain her 0.6BTC in the channel. Bob, after waiting, can then sign his revocable delivery output to acquire his 0.4BTC in the channel. Note that Alice cannot sign the breach

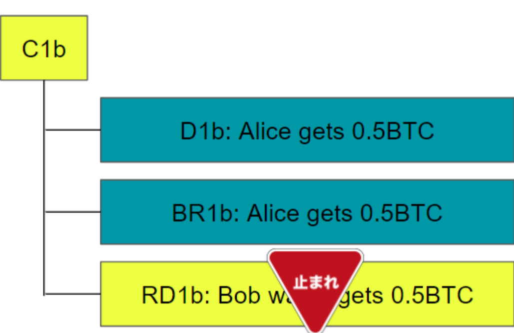

Figure 4. An incorrect channel closing, referring to the state of the channel as described in figure 5

rememdy output for the state 2 commitment transaction because Alice does not yet know Bob's secret for state 2.

In figure 4, we see an example of an incorrect commitment transaction broadcasted by Bob. Because state 1 is not the most up to date state, Bob is effectively being dishonest and thus Alice can acquire all funds in the channel. To do this, Alice can sign her delivery transaction to acquire the 0.5BTC that is hers at state 1. Alice can sign the breach remedy output before Bob can sign the revocable delivery output (recall that these are, in fact, the same output) to acquire the 0.5BTC that was Bob's at state 1. Bob's revocable delivery output will now be registered as an invalid double-spend. Alice is able to use the breach remedy because she learned Bob's secret for state 1 when they agreed upon state 2.

To advance to a new state i, Alice and Bob must do the following:

1. Alice and Bob agree on the values in both parties' accounts for state $i+1$. Alice and Bob both generate signatures for the other's commitment transaction (i.e. Alice signs Cib and Bob signs Cib) and send these.
2. Alice and Bob exchange their secrets from the previous state.

3.3. Signatures

Each transaction to be broadcasted must be signed by both parties. All transactions use unique public key and secret keys. With n states in the channel, we must know the breach remedy transaction secret for all n previous states in the channel, but we wish to need to keep fewer than n signatures. To achieve this, the Bitcoin Lightning network uses hierarchical deterministic wallets, which are effectively reverse Merkel trees, that can reduce the number of keys kept by a party [4]. Effectively, the parents in the trees are the keys of the latter states. By knowing the key for state i, one can generate its child, the key for state $i-1$ or $i-2$, by hashing the key and concatenating a constant parameter.

4. Probabilistic Payments

Probabilistic payment systems were first developed by Rivest and Micali in 2002 [5] as a way to efficiently handle micropayments. Micropayments are very small transactions. Depending on the context, this can mean transactions on the order of dollars, cents, or amounts even smaller than a cent. Micropayments can be useful for buying music,

watching movies, or using pay-per-view websites. However, these transactions can be unrealistic if the overhead on each transaction is too large relative to the amount of money exchanged.

The probabilistic payment system, referred to as Peppercoin in the original paper [6], was originally designed to mitigate relatively large fees associated with credit card microtransactions. The general idea of probabilistic payment is that transactions only have some low probability of going through, but they have a high payload if they go through. So instead of Alice paying Bob 1¢, she pays Bob $1 with probability $1/100$. Suddenly, they will only be paying the 5¢ fee once for every 100 transactions. Thus probabilistic payments do the job of bundling together micropayments to minimize the amount of overhead work that must be done.

One of the biggest benefits of the Lightning Network is that it helps people do smaller and more frequent transactions. However, the size of transactions possible through Lightning are bounded below by the smallest unit of Bitcoin: the Satoshi. One way to enable sub-Satoshi transactions would be to allow Alice to send Bob Satoshis with some probability. Implementing a probabilistic payment system in Lightning would enable the network to be able to handle "true micro-payments.

Because of security restrictions native to the Bitcoin network, only a limited number of functions (called opcodes) can be used by the miners to determine the outcome of the transaction. As a result, the probabilistic aspect of our probabilistic payment system relies on the lengths of the pre-image of hashes, instead of properties that may be more intuitive.

Here are the steps that go into how Alice would pay Bob half a Satoshi under the probabilistic payment infrastructure in Bitcoin Lightning. h is a hash function.

1. Bob chooses two numbers, Y_1 and Y_2 and sends the hashes to Alice.
2. Alice picks a random number X. It should be 20 or 21 bits long.
3. Alice commits to X by sending $h(X)$ to Bob, along with two transactions (both half-signed by Alice). See below for the format of the transactions.
4. Bob has several options at this point:
   (a) Sign both and broadcast both
   (b) Sign neither
   (c) Sign Transaction N (where $N = 1$ or 2), and broadcast
   (d) Send the signed Transaction N and Y_N to Bob

Transaction 1:

- Sig_A and $len(X) = 20$ (Bob gets 0S)
- Sig_B and Bob waits (Bob gets 1S)
- Sig_A and Y_2 (Bob gets 0S)

Transaction 2:

- Sig_A and $len(X) = 21$ (Bob gets 0S)
- Sig_B and Bob waits (Bob gets 1S)
- Sig_A and Y_1 (Bob gets 0S)

What happens next depends on what Bob picks:

(a) If Bob broadcasts both, the miners pick which transaction to carry out. This is essentially the same as signing and broadcasting a single transaction.

(b) If Bob broadcasts neither, Alice can close the channel after waiting some time. Obviously, Bob does not get any money.
(c) If Bob signs and broadcasts Transaction 1, that means Bob is betting that the $len(X) = 21$. He gives Alice a window of time to prove that he picked wrong. If Alice reveals that $len(X) = 20$, then the transaction goes through where Bob gets 0S. If nothing happens, then Bob gets 1S. A very similar thing happens if Bob broadcasts transaction 2.
(d) Bob picks this option when he is planning on keeping the channel open for more future transactions, and isnt ready to broadcast the channel state to miners yet. We expect this feature to be important because of the importance of high frequency transactions for the Lightning Network. Lets say that Bob picks Transaction 1 (betting that $len(X) = 21$). The transaction must be resolved before it is broadcasted, because otherwise the channel funds between Alice and Bob may be depleted without them realizing. Because of that, Bob must be forced to commit to a transaction when he decides. When he sends a signature of the fully signed Transaction 1 to Alice, he also sends her Y_1. If he decides to broadcast Transaction 2, he wont be able to get any money because Alice can sign that transaction with her knowledge of Y_1. With this commitment scheme, they are able to continue making un-broadcasted transactions until they close the channel.

This schema can be extended to accommodate probabilities even smaller than $1/2$. In general, for $1/n$, Alice can transmit n different transactions, each with different possibility for the length of X. Bob would generate $Y_1, \ldots, Y_n$. When Bob commits to Transaction i, he sends $\{Y_j\}_{j \neq i}$ to Alice. For increasingly large n, however, this schema becomes increasingly complex and inefficient.

5. Implementation

5.1. Existing Code

The code for Lightning Network is written entirely in Go. Users can fund payment channels and create transactions on the Testnet using a client in the terminal. The terminal is a Bash-like interface with commands like *help* and *ls*. Users can type *ls* (Figure 5) to see a summary of their wallet, including connected peers, channel history, past transactions, addresses, and account balance.

There are two ways to send money to a peer through the Lightning network. The first is the same as how a user would send Bitcoin through the regular Bitcoin network. The receiver generates a new address (tb1q...) for one-time use and gives it to the sender. In this client, the sender can enter ”send tb1q... [amount]”. The transaction is broadcasted and after a few minutes, a miner adds a block to the public ledger and confirms the transaction. A disadvantage to this method is that it may take a while for funds to become available for use.

The other method, unique to the Lightning Network, is to open a payment channel, say, between Alice and Bob. Alice has to know Bob’s public key (ln...), and can connect to Bob using ”con ln...:[port]”. Note that unlike the random tb1q addresses, this connection is using the identifiable public key. Under peers, Alice can find the ID of her connection with Bob, and fund a channel using ”fund [ID] [capacity] [initialSend]”. Alice

Figure 5. A screen capture of a sample Lightning client. Users can type "ls" to view a summary of their wallet, past transactions, connections, and channel statuses.

and Bob can use this payment channel to exchange money back and forth using "push [channelID] [amount]" for immediate use. Only when they close the channel are the final amounts broadcasted and added to the public ledger.

5.2. Probabilistic Payment Implementation

We extended Lightning's API so that we could make probabilistic payments in the channel using "push [channelID] [amount] [odds]". The command probabilistically pushes the given amount (in satoshis) to the other party on the given channel with probability $1/odds$. The code written for this implementation can be found in the Appendix.

6. Vulnerabilities

As the Lightning Network is still under development, there are still many bugs in both the implementation and theory of the system.

6.1. Implementation Vulnerabilities

1. **Address Format:** Bitcoin and its variants use Base58Check encoding to generate addresses for receiving payments. The encoding process involves hashes, check-

sums, and appending an address prefix unique to the relevant blockchain-based currency. Testnet coin uses the prefix ”tb1q”, but due to a bug in the code, the encoding process only appended ”tb1.” Since the addresses were incorrectly formatted, all the transactions sent to these addresses were rejected by Testnet, and the associated money disappeared. This vulnerability would not be easily identified by inexperienced users. While the prefix has since been fixed, the root of the problem could be even better remedied if Lightning implemented a verification system for address formats before broadcasting transactions to the Testnet.

2. **Network Issues:** Creating a payment channel required establishing a connection to another Lightning node with a static IP address. The Lightning repository currently only supports Windows and Linux, so many of us had to run the code on a virtual machine. The process of configuring static IPs on virtual machines is rather complicated, and limited us from using some of Lightning’s features. Acquiring a static IP on MIT’s campus also requires either using a wired connection or going through a complicated approval process with IS&T [8]. This is more of a usability issue than a vulnerability.

6.2. Theoretical Vulnerabilities

1. **Exhausted Channels:** Consider a payment channel between Alice and Bob. An exhausted channel is when one party, say Alice, has completely run out of funds. Normally, the only thing incentivizing a party not to publish an old channel state is if the other party catches them, they can utilize the breach remedy condition and cause the guilty party to lose all the money in the channel. In this case, since Alice has nothing to lose, nothing is stopping her from publishing an old state in which she has more money. If Bob does not notice this in time, he will only get the money from the old channel state. Currently, the only way to prevent this attack is by making sure channels are never exahusted.
2. **Segregated Witness:** The SegWit soft-fork introduces fixes to transaction malleability and increases Bitcoin’s block size limit. SegWit is set to activate as soon as it reaches 95% support, but it is currently hovering at around 33% due to mostly ideological opposition [7]. By solving malleability, SegWit allows the wide scalability of second-layer Bitcoin networks like Lightning. Otherwise, every Lightning client would have to run a full node in order to be completely trusted.
3. **Anonymity:** Lightning requires a party to open payment channels using the other party’s identifiable public key. This is the same public key used to send messages through the terminal interface. Unlike addresses, public keys are not one-time use and cannot be infinitely-generated.

7. Conclusion

As the first decentralized digital currency, Bitcoin’s very existence changes the way we think about money and economy. Bitcoin has been enjoying rapid growth since it was released in 2009. But now that same growth is causing it to struggle to accommodate today’s number of users and different kinds of transactions that it was not originally in-

tended for. Bitcoin Lightning is an attempt to remedy that, with a system that accommodates more high-frequency and low-value transactions and reduces the load on Bitcoin servers.

In this paper, we evaluated the Bitcoin Lightning system, with a special emphasis on security. We found implementation bugs and system flaws, such as a lack of incentive against broadcasting past states when the payment channel has been exhausted. Another flaw in the system is the lack of total anonymity, because activity is tied to a public key. This also reflects the lack of total anonymity in the Bitcoin system.

The main focus of our project was to evaluate the possibility of, and implement, a probabilistic payment system to Bitcoin Lightning. Probabilistic payment would extend the focus of Lightning to not only allow high-frequency low-value payments, but also true microtransactions with values of less than a Satoshi. We implemented a rudimentary version of it that allows two people to carry out a probabilistic payment within a Lightning channel using the protocol described in section 4. One of our next steps include fixing errors coming from deadlocking, and implementing all of the security checks that our protocol relies upon to be safe.

Bitcoin Lightning is still a very new addition to Bitcoin, having been published around a year and a half ago. It hasn't been incorporated into the main infrastructure yet, with one major reason being that many of the features have not been ironed out yet. We believe the system has the potential to bring Bitcoin to a higher level of scalability and usability as it needs it right now. Furthermore, if Lightning incorporates probabilistic payment, it could change the environment around micropayments and make possible a new class of transactions.

References

[1] Poon, J., & Dryja, T., The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments, Jan. 2016

[2] MIT Digital Currency Initiative, Lit, (2017), GitHub repository, https://github.com/mit-dci/lit

[3] Satoshi Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, Oct 2008.

[4] Wuille, P., BIP 0032: Hierarchical Deterministic Wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, Feb 2012.

[5] Micali S., Rivest R.L. (2002) Micropayments Revisited. In: Preneel B. (eds) Topics in Cryptology CT-RSA 2002. CT-RSA 2002. Lecture Notes in Computer Science, vol 2271. Springer, Berlin, Heidelberg

[6] Rivest R.L. (2004) Peppercoin Micropayments. In: Juels A. (eds) Financial Cryptography. FC 2004. Lecture Notes in Computer Science, vol 3110. Springer, Berlin, Heidelberg

[7] Marshall, A. Segwit, Explained. https://cointelegraph.com/explained/segwit-explained, Apr 2017.

[8] How can I request a static IP Address and host name? http://kb.mit.edu/confluence/pages/viewpage.action?pageId=4982482

Appendix

The code written for this project can be accessed here: https://github.com/jbloxham/lit.

The full repository for the Bitcoin Lightning network can be accessed here: https://github.com/mit-dci/lit.

Acknowledgements

We would like to acknowledge Prof. Ron Rivest, Prof. Yael Kalai, and the 6.857 Computer and Network Security Teaching Assistants for their support in this project. We would also like to extend our thanks to Tadje Dryja and the MIT Digital Currency Initiative in their collaboration on the project.