The Lightning Network Watchtower

Joel Davidson[1]**, Tanner Lillich**[1]**, and Elsa Velazquez**[1]

[1]CU Boulder, CSCI 4830

Abstract
Bitcoin Blockchain is facing its next revolution– scalability, as the Lightning Network (LN) implements the second layer protocol that uses a network of micropayment channels, a more malleable bidirectional payment route that uses a different sighash type, and a series of decrementing timelocks [1]. As LN offers a Bitcoin scalability solution, it brings with it a new set of problems [2]. In our project, we address the need for the constant vigilance of a node being online to protect against specific attacks [2]. Throughout our project, we encountered numerous issues that were likely the result of "that the [LN] software is still in beta," even as of January 2019 [2]. Ultimately, our conclusion is that in its present state, the best solution is to have a 3rd party Watchtower which would watch the blockchain for fraudulent transactions and automatically broadcast a revocation transaction if it finds one.

Introduction. The motivation behind our project was to address some of the potential issues with the Lightning Network (LN). LN provides greater privacy and scalability than on-chain transactions. Some issues yet to be addressed are: needing to generate an invoice prior to a payment, both parties needing to be online for a payment to go through, the capacity of all nodes on the path from the payer to the payee must have at least as much funds on them as the payment amount or the payment will be dropped. Sphinx send mode for spontaneous payments [3] solves the need for generating invoices. Atomic mulit-path routing (AMPP) solves the problem of needing capacity on all the nodes [4]. There is not currently a solution to the issue of both parties having to be online for a payment to go through. That is the nature of the lightning network though. If a node is offline, not only can it not receive payments, but it can't route payments either. So, the point of the network is for routing nodes to try to stay up as much as possible. Individual users can come and go as they please. The problem we attempt to solve is the problem of preventing double spend while away from the network. In the bitcoin blockchain, there is a scalability problem, and limited transaction capability. The bitcoin blockchain can handle very few transactions in comparison to what fiat currency services such as VISA can handle. Visa can conduct as much as 50,000 transactions per second. Bitcoin can only conduct 5-7 tx/sec on chain. The LN is a scalability solution for Bitcoin that moves transactions off chain. It can conduct unlimited off chain transactions. We propose to solve for the problem of LN vigilance by means of a "Watchtower." A watchtower is a third party server that constantly watches the blockchain for fraudulent transactions and if it finds one, broadcasts the revocation transaction to the fraudulent commitment transaction. A simpler solution would be an alert system that would watch the blockchain for any transaction from the multisig address that holds the funds of the lightning channel and sends an email to the parties involved. The implementation attempts depended on the version of the LN as well as working around the already built architecture. The LN transactions are described in the next section and lay the groundwork as the driving mechanisms for our proposed solutions. Our solutions were greatly influenced by the programming language they were implemented in, hence we have created two versions of procedures and outcomes when appropriate, to fully address potential variations.

2a Background on LN Transactions. There exist a multitude of resources that describe the LN transactions, which include the Basics of Lightning Technology (BOLT) [5] and the lightning network repo [6]. Here, we use the Lightning Network Whitepaper descriptions which aligned well with the BOLT and GitHub descriptions [1]. There was a clear overall similar intent and effect in both programming language versions, but there was a concrete variation in code implementation, depending on the language version. We will specifically address the multidirectional, two party transactions, as this is what our subsequent attacks, described in section 2c, and project findings were based upon.

2a1 Hashed Time Lock Contracts. Hash Time Locked Contracts (HTLC's) indicate a special 'locked out' time type of smart contract. There are two possible cases for closing a channel. The first is a mutual or "cooperative" close. This is a final settlement transaction, signed by both parties, which instantly receive their funds. This case makes it unnecessary to submit an intermediate commitment transaction. The other case is a unilateral close. In this case, one party (Alice) posts a commitment transaction to the blockchain. The other party (Bob) immediately receives his funds. But Alice has a HTLC against her for the period of 1000 mined blocks, not allowing her to receive her funds until then. This HTLC provides Bob with an opportunity to submit a revocation transaction. The purpose of this would be to show that there was a more recent transaction than that which Alice published. We need the unilateral close option because one of the parties could disconnect, but this is possible to exploit. If Alice submits to the blockchain an earlier commitment transaction that has more funds on her side of the channel, that is stealing form Bob.

2a2 Multi-hop payments. The lightning network is a network of bi-directional payment channels. By linking the payment channels, a node can transmit funds to another node that it is not connected to as long as there is a path of connected

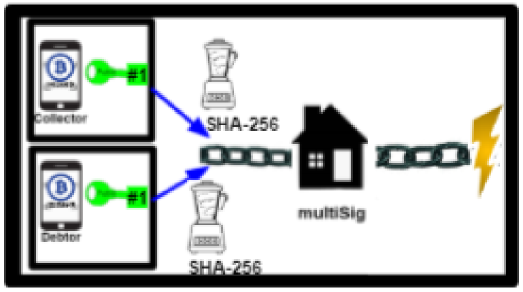


Fig. 1. Figure 1- Step 1 of Multidirectional Micropayment LN Channel
The initial step in creating a payment channel on the LN is a funding transaction ("anchor transaction"). This transaction is transmitted to the blockchain and mined in order to start the channel. The funding transaction is a transaction from the party that is opening the lightning channel. It puts funds into a 2 of 2 multisig address and gives both parties the keys to spend this output to the original party.

nodes and they all have funds on them equal to the transaction being sent[1].

2b Multidirectional multiSig as Micropayment Channels Using Cryptographic Signatures. Per section 3.3.5 of the LN Whitepaper, a party can fund and open a channel as an address on the Bitcoin main-chain, as balances are updated n times as necessary on the LN side-chain, with only the final balances broadcast onto the blockchain main-chain [1]. The following is a step-by-step description of the LN in action [1].
Step 1. Opening a channel is done by the LN creating an address on the main-chain that requires both parties' signatures, and uses the SIGHASH NOINPUT script.

- Both parties agree to create a Commitment Transaction
- Together they create one multiSig
- Each party puts in their Public Key of their asymmetric key pair
- The funds function an a escrow to fund the channel and can't be spent
- The collector must match the debtor's UTXO amount (if the debtor skips out, the main-chain is still correctly balanced by the collector'sescrow)

Commitment Transaction 1 = multiSig(Collector's Public Key 1, Debtor's Public Key 1)
Step 2. A new Commitment Transaction is created on the side-chain each time the debtor's tab changes. Both parties must sign the same Revocable Sequence Maturity Contract (RSMC) with their private keys. The RSMC is also used to set the timelock (the predefined number of main-chain blocks that must be published before the channel is closed), and uses the OP RELATIVECHECKLOCKTIMEVERIFY script.

- The 'tab' changes so the UTXO changes
- A new UTXO requires a new Commitment Transaction
- For security reasons (to avoid replay attacks based on merkle trees property) they have to use new key pairs

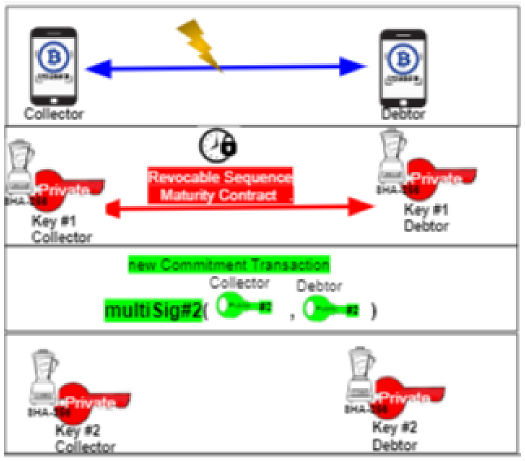


Fig. 2. Figure 2- Step 2 of Multidirectional Micropayment LN Channel
Create new commitment transactions on the LN.

- Using their private keys both parties sign the same RSMC
- New key pairs are generated for each party by adding
- +1 to each party's public key
- The current and verifiable key pairs are the new keys: Public Keys 2
- Commitment Transaction 2 = multiSig(Collector's (Public Key 2, Debtor's Public Key 2)
- New private keys are generated as a result of the new public keys (a features of asymmetric encryption)
- Both this new tab balance and the previous tab balance (the UTXO's) are broadcast on the sidechain channel only (the Lightning Network)

It is important to note that off-chain, the tab can change an infinite number of times before the time lock expires, there are no main-chain fees for any of these changes while on the sidechain, and the updates happen only on the sidechain.
Step 3. It is in this next step that if both parties do not close cooperatively, the Watchtower comes into play and and enacts punishment for bad actors. The specific attack is described in practical detail in the next section, and is described here per the specifications of the LN Whitepaper [1].

- In the case of a problem, the following protocol is enacted by the LN [1]:
- The channel is closed and published on the mainchain
- The proof is based on timestamps- the most current transaction will have the latest timestamp
- The penalty: the bad party loses ALL their funds to the victim party
- The victim has to wait until the timelock expires (specified in the in the RSMC) to get those funds

Any of these 3 bad actors are punished by the LN, and the usefulness of the Watchtower applies to any of the cases in Figures 3-5.

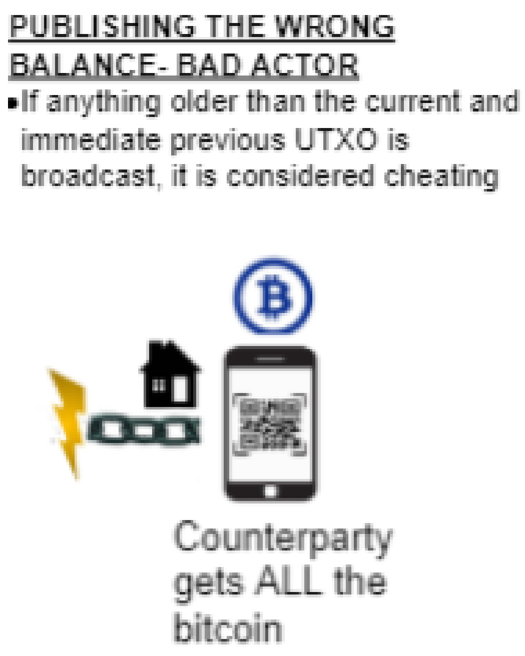

PUBLISHING THE WRONG BALANCE- BAD ACTOR
•If anything older than the current and immediate previous UTXO is broadcast, it is considered cheating
Counterparty gets ALL the bitcoin

Fig. 3. Figure 3- Punishable Problem 1 on LN A bad actor publishes the wrong balance.

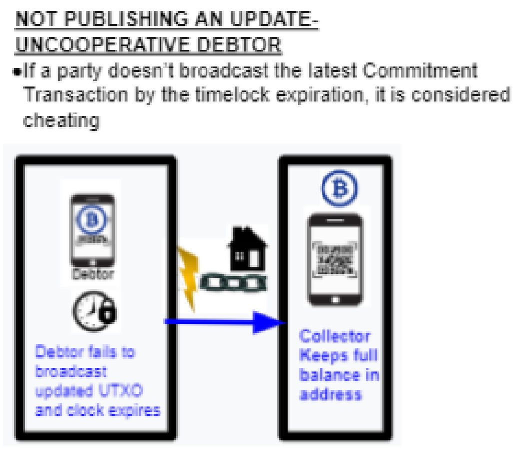

NOT PUBLISHING AN UPDATE- UNCOOPERATIVE DEBTOR
•If a party doesn't broadcast the latest Commitment Transaction by the timelock expiration, it is considered cheating
Debtor
Debtor fails to broadcast updated UTXO and clock expires
Collector Keeps full balance in address

Fig. 4. Figure 4- Punishable Problem 2 on LN An uncooperative debtor won't close the channel.

2c Multidirectional Multisig As An Attack. In our use case, we chose the specific case of Alice attempting to frame Bob as shown in Figure 5, where Bob would appear to commit punishable problem 3 and lose all his money to Alice. Her attack was based in letting the hashed timelock elapse and then publishing a previously revoked transaction, with Bob being unaware that his node had gone offline and he would lose the opportunity to submit a more recent transaction and in turn be the one that gained all the funds from Alice as the punished party. In this case, the Watchtower would have alerted Bob so he could act accordingly in time.

2d Background on LN Potential Attacks. The use case attack we chose is the classic bitcoin Double Spend Attack [7]. A double spend attack on the lightning network would consist of a party submitting to the blockchain a commitment trans-

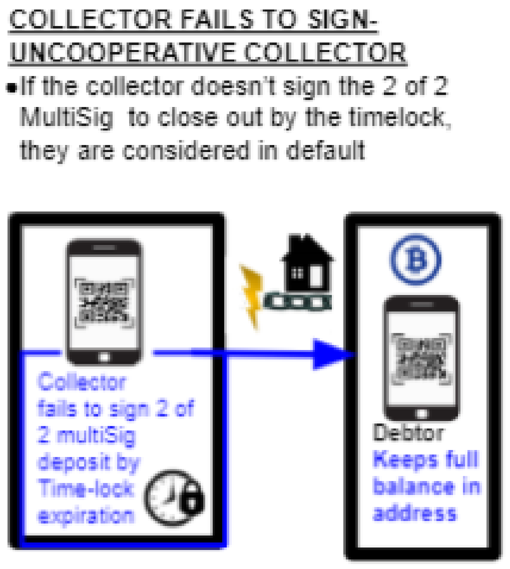

COLLECTOR FAILS TO SIGN- UNCOOPERATIVE COLLECTOR
•If the collector doesn't sign the 2 of 2 MultiSig to close out by the timelock, they are considered in default
Collector fails to sign 2 of 2 multiSig deposit by Time-lock expiration
Debtor Keeps full balance in address

Fig. 5. Figure 5- Punishable Problem 3 on LN Uncooperative collector.

action that is not the most recent transaction and is of a state where the stealing party has more funds on their side of the channel. Unilateral closes with HTLCs mean that the victim has the chance to broadcast the revocation of this fraudulent transaction and sweep all the funds from the channel to their side which punishes the thief [8].

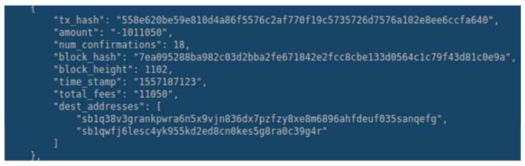
2e The Problem Of Countering Said Attack. The problem witin the LN structure is that the victim's node must be online to detect and then transmit this revocation transaction. Currently, there is no way for the victim to have any indication that they are vulnerable.

2f The Watchtower As a Solution for Said Attack. The victim can solve for the attack by having someone else watching for them. A program that maintains the most up to date state of a channel and watches the blockchain for a fraudulent commitment transaction on behalf of the victim is called a "watchtower". Watchtowers exist in the LN implementations but they are a part of the LN node and are only operating if the node is online. For this project, we refer to a watchtower as a third party that would watch for a node that is offline, or that would alert a specific user for activity on a specific channel. Third party watchtowers don't exist yet, and the different approaches to Watchtower solutions stemmed from the different implementations in the C and Go/Golang versions of the LN .

3 Technical Description of Our Proposed Solutions for the LN Problem of Vigilance. We will propose two potential solutions for the LN that solve the problem of vigilance. There are two versions of the LN currently available. Each version is at a different implementation stage and therefore offers slightly different problems and therefore slightly different solutions.

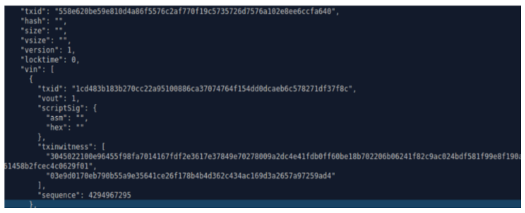
3a The C Version Implementation. We looked into implementing a watch tower and adding this to one of the imple-

JD, TL, EV *et al.* | LN Watchtower bioRχiv | 3



```
    {
        "tx_hash": "558e620be59e810d4a86f5576c2af770f19c5735726d7576a102e8ee6ccfa640",
        "amount": "-1011050",
        "num_confirmations": 18,
        "block_hash": "7ea095288ba982c03d2bba2fe671842e2fcc8cbe133d0564c1c79f43d81c0e9a",
        "block_height": 1102,
        "time_stamp": "1557187123",
        "total_fees": "11050",
        "dest_addresses": [
            "sb1q38v3grankpwra6n5x9vjn836dx7pzfzy8xe8m6896ahfdeuf035sanqefg",
            "sb1qwfj6lesc4yk955kd2ed8cn0kes5g8ra0c39g4r"
        ]
    },
```

Fig. 6. Figure 6- LNCLI Chain Transactions Create new commitment transactions on the LN.



```
  "txid": "558e620be59e810d4a86f5576c2af770f19c5735726d7576a102e8ee6ccfa640",
  "hash": "",
  "size": "",
  "vsize": "",
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "1cd483b183b270cc22a95100886ca37074764f154dd0dcaeb6c578271df37f8c",
      "vout": 1,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "txinwitness": [
        "3045022100e96455f98fa7014167fdf2e3617e37849e70278009a2dc4e41fdb0ff60be18b702206b06241f82c9ac024bdf581f99e8f190a
51458b2fcec4c0629f01",
        "03e9d0170eb790b55a9e35641ce26f178b4b4d362c434ac169d3a2657a97259ad4"
      ],
      "sequence": 4294967295
    },
```

Fig. 7. Figure 7- Spent Transactions Output of spent transaction command.

mentations of the LN code. The watchtower would have the capability to automatically submit the most recent revocation transaction in the case that the commitment transaction posted by the other party was not the most recent state of the channel. There are a few ideas of how this would work. The most likely idea is that it would be a node on the LN and when one party sends a payment to another, they would also send this updated state to the watchtower and maybe a few satoshis to perform the watchtower function. Our next plan was an alert system. An alert could be a separate program from the LN node. It would monitor a bitcoin node for a transaction that spent the funds from the funding transaction multi-sig address. It would run as a crontab script that would pull all transactions from the address and if it found it was being spent, it would send an email to the party that was off line. We created some of the main components of an alert program as described below.
In order to monitor a LN channel, we took the following steps (using the LND LN client).

1. After creating a LN channel, find the address that the funds are locked in. This is a transaction on the blockchain and can be found by running lncli listchaintxns.
2. btcd needs to be running with the –txindex flag set
3. Run the command *btcctl* searchrawtransactions "address" with the address from the funding transaction. This will give the transactions for that address and can be used to determine if the funds have been spent.

In order to monitor a LN channel, we took the following steps (using the LND LN client). After creating a LN channel, find the address that the funds are locked in. This is a transaction on the blockchain and can be found by running *lncli* listchaintxns.

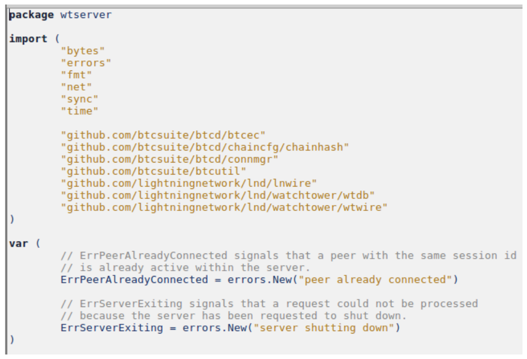
3b The Solution for the Go Version . A possible similar implementation attempt in the Go version of the LN was to send alerts each time a channel was force closed by a party sharing the same channel. The major difference in the Go/Golang version of the LN is that the Go version has already begun implementing watchtower features. While not yet fully functional as described in the white-paper [LN Whitepaper], the present features make coding our solution for a node-based watchtower implementation more feasible, but implementing a solution in Go/Golang creates its own set of challenges. The challenges are discussed in section 4.b. Go/Golang Implementation Challenges, of this paper.

3c Other Watchtower Solutions Currently Being Implemented. Lightning Labs and Blockstream are fully established software engineering firms [9]. They are responsible for the development and implementation of the majority of the code.

4 Challenges In Solution Implementations. Our first step was to set up lightning node. This process proved to be extremely difficult, specifically due to hard to follow and inaccurate instructions online, and bugs in the existing code. It took a long time in order to correctly install these nodes onto our computers. We were eventually able to successfully install and run lightning nodes, open channels, make payments from one to another node and close channels on simnet. During this process we created an installation tutorial that is much easier to follow and that works on our machines running Linux and MacOS. This full tutorial is attached in the appendix. We hope that this tutorial will allow more people to join the LN community, because we worked to make our instructions easy to follow.

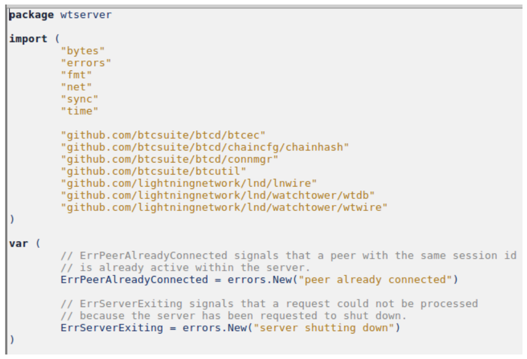
4a C Language Implementation Challenges. It took a great deal of time to simulate a double spend attack on the lightning network. There are no RPC commands that allow a user to commit fraud with the LN client implementations that are available currently on the C version of LN. The intention was to make a patch to the code that would send a revoked commitment transaction to the blockchain. We looked through the LND C-Lightning implementation to try to find how to submit a revoked commitment transaction but did not find a method. Creating accounts on the IRC channels to ask developers questions as well as bitcoin.stackexchange.com also did not yield solutions and although a few questions were answered, no one was able to tell us how to simulate this attack.

4b Go/Golang Implementation Challenges. The most significant challenge in testing solutions with the Go/Golang code implementation were with the use of the Go language. Because the make clean and make commands on a Go project work differently than typical projects, even the use of branched gitHub pages and code editors such as Visual Studio Code proved to be cumbersome and difficult. The other challenge was that in using the Simnet version, it was not possible to use the Blockchain API to access live data and therefore also found it impossible to broadcast previously revoked transactions deliberately, in order to test our use cases to the fullest extent. It was possible, however, to simulate broadcasting an attack by means of closing a channel by force, which proved to be a good use case for why the Watchtower



```
package wtserver

import (
	"bytes"
	"errors"
	"fmt"
	"net"
	"sync"
	"time"

	"github.com/btcsuite/btcd/btcec"
	"github.com/btcsuite/btcd/chaincfg/chainhash"
	"github.com/btcsuite/btcd/connmgr"
	"github.com/btcsuite/btcutil"
	"github.com/lightningnetwork/lnd/lnwire"
	"github.com/lightningnetwork/lnd/watchtower/wtdb"
	"github.com/lightningnetwork/lnd/watchtower/wtwire"
)

var (
	// ErrPeerAlreadyConnected signals that a peer with the same session id
	// is already active within the server.
	ErrPeerAlreadyConnected = errors.New("peer already connected")

	// ErrServerExiting signals that a request could not be processed
	// because the server has been requested to shut down.
	ErrServerExiting = errors.New("server shutting down")
)
```

Fig. 8. Figure 8- Node Shutting Down
The Go implementation currently is structured to alert users when a node is shutting down, however it is not yet effectively implemented.



```
package wtserver

import (
	"bytes"
	"errors"
	"fmt"
	"net"
	"sync"
	"time"

	"github.com/btcsuite/btcd/btcec"
	"github.com/btcsuite/btcd/chaincfg/chainhash"
	"github.com/btcsuite/btcd/connmgr"
	"github.com/btcsuite/btcutil"
	"github.com/lightningnetwork/lnd/lnwire"
	"github.com/lightningnetwork/lnd/watchtower/wtdb"
	"github.com/lightningnetwork/lnd/watchtower/wtwire"
)

var (
	// ErrPeerAlreadyConnected signals that a peer with the same session id
	// is already active within the server.
	ErrPeerAlreadyConnected = errors.New("peer already connected")

	// ErrServerExiting signals that a request could not be processed
	// because the server has been requested to shut down.
	ErrServerExiting = errors.New("server shutting down")
)
```

Fig. 9. Figure 9- Attempting To Rebroadcast A Revoked Commitment Transaction
The Go implementation does not yet allow for a user to broadcast a revoked transaction, however this showcases a possible attack attempt that is punishable according to the LN Whitepaper. It is possible that a watchtower could send an alert to a potential victim when the broadcast function is made available to users.

is necessary, as Alice was effectively able to steal all of the funds from the channel. The code snippets in Figures 8 and 9 are taken from the LN implementation and show possible locations where an alert could be set to trigger an email. There are other possible points of Watchtower implementation. Many of these include 3rd party reliance, however there are also some solutions that could be attempted by each individual node. Ultimately, the solution would include a peer to peer, or selective peer, community watchtower.

5 Further Research. We partially initiated our further research, which was to implement an alert system. An alert could be a separate program from the LN node. It would monitor a bitcoin node for a transaction that spent the funds from the funding transaction multi-sig address. It would run as a crontab script that would pull all transactions from the address and if it found it was being spent, it would send an email to the party that was off line. We created some of the main components of an alert program as described below.

5a Next Steps. It is our intention to continue working to contribute the useful Watchtower to the LN community. Currently, the largest transactions are of 400 Satoshis so there is not as much interest in the topic, but as the scalability issue is resolved and more people use LN, the Watchtower will prove to be useful and well appreciated.

6 Floating Figures and Lists.

7 Appendix of Compiled Resources.

http:https://github.com/elsaVelazquez/crypto/blob/master/LNsetup

Acknowledgements

Dr. Eric Wustrow, CU Boulder Cryptocurrencies Securities Professor

Bibliography

1. Joseph Poon, Thaddeus Dryja (2016) *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*, http://lightning.network/lightning-network-paper.pdf.
2. Larry Cermak (2019) *The growth of the Lightning Network has been remarkable. But there's a catch.*, http://www.theblockcrypto.com/2019/01/15/the-growth-of-the-lightning-network-has-been-remarkable-but-theres-a-catch/.
3. Roasbeef (2019) *[WIP] multi: add new draft sphinx send mode for spontaneous payments 2455*, http://github.com/lightningnetwork/lnd/pull/2455.
4. Olaoluwa Osuntokun (February 2018) *[Lightning-dev] AMP: Atomic Multi-Path Payments over Lightning*, http://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.
5. Rene Pickhardt and Roasbeef (February 2018) *Lightning Network Specifications*, http://github.com/lightningnetwork/lightning-rfc.
6. lightningnetwork (September 2018) *lnd Lightning Network Daemon*, http://github.com/lightningnetwork.
7. Satoshi Nakamoto (n.d.) *Bitcoin: A Peer-to-Peer Electronic Cash System*, http://bitcoin.org/bitcoin.pdf?.
8. ariard (February 2019) *BOLT 3: Bitcoin Transaction and Script Formats*, http://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md.
9. Olaoluwa Osuntokun Roasbeef (2019) *Olaoluwa Osuntokun Roasbeef*, http://github.com/github.com/Roasbeef.

JD, TL, EV *et al.* | LN Watchtower bioRχiv | 5