6-27-2023

Enabling Third Layer Bitcoin Applications Using Lightning Network

Ahmet Kurt
Florida International University, akurt005@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

ENABLING THIRD LAYER BITCOIN APPLICATIONS USING LIGHTNING NETWORK

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL AND COMPUTER ENGINEERING

by

Ahmet Kurt

2023

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Ahmet Kurt, and entitled Enabling Third Layer Bitcoin Applications using Lightning Network, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

A. Selcuk Uluagac

Alexander Perez-Pons

Mohammad Ashiqur Rahman

Hemang Subramanian

Kemal Akkaya, Major Professor

Date of Defense: June 27, 2023

The dissertation of Ahmet Kurt is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2023

DEDICATION

To my family.

ACKNOWLEDGMENTS

First and foremost, I want to express my deepest appreciation to my advisor Prof. Kemal Akkaya for his continuous guidance and unwavering support throughout my journey as a doctoral student. Whenever I lost hope, he has motivated me to keep going. I would not be able to write this dissertation without his mentorship, expertise and help. I want to extend my appreciation to Dr. A. Selcuk Uluagac as well. His insights and suggestions were crucial during our collaborations.

This dissertation would not be possible without the help of my collaborator and friend Enes Erdin. He has a brilliant mind and working with him contributed to the way I approach challenging research problems. I should also acknowledge the help of my collaborators Dr. Suat Mercan, Dr. Mumin Cebe, Hadi Sahin and Ricardo Harrilal-Parchment for their contributions.

I also appreciate Dr. Alexander Perez-Pons, Dr. Mohammad A. Rahman and Dr. Hemang Subramanian for serving on my dissertation committee and generously giving their time to advice and help.

I feel blessed to be surrounded with great friends at Advanced Wireless and Security Lab (ADWISE). The time I spent during my Ph.D. would not be as enjoyable without my friends Yacoub, Yassine, Harun and Maryna.

I would like to thank Florida International University Graduate School for awarding me with the Dissertation Year Fellowship on my last year of Ph.D. Without this fellowship, I would not be able to complete this dissertation.

ABSTRACT OF THE DISSERTATION

ENABLING THIRD LAYER BITCOIN APPLICATIONS USING LIGHTNING NETWORK

by

Ahmet Kurt

Florida International University, 2023

Miami, Florida

Professor Kemal Akkaya, Major Professor

When Bitcoin was introduced in 2009, it created a big sensation in the world as it was first of its kind. Since then, a lot of different cryptocurrencies were proposed. Today, cryptocurrencies can be used to pay for goods and services similar to using cash or credit cards. However, none of them could replace or supersede Bitcoin in usage or market capitalization. Current market conditions still imply that it will stay the same way. However, Bitcoin suffers from very low transaction per second (TPS) which limits its usability on large scale. There have been numerous proposals to increase its scalability such as *block size increase*, *Schnorr signatures*, *side chains* and *layer-2 networks*. Among all, layer-2 networks is by far the most promising solution as shown with the success of the Lightning Network (LN) which grew exponentially over the years reaching 16,000 public nodes worldwide.

LN was implemented in 2017 with the aim of decreasing the load on the Bitcoin blockchain by facilitating the transactions on its decentralized network which enables almost free and instant Bitcoin payments. It works by processing the payments *off-chain* meaning payments are not recorded on the Bitcoin blockchain. In order to transact on LN, users need to open at least one LN channel to one of the nodes in the network in advance and put some funds in the channel. Emergence of LN opened new doors to many potential novel applications that can utilize its infrastructure.

Indeed, LN's underlying network offers a perfectly covert communication medium to enable security and privacy by default. This creates opportunities for the sake of good and bad. *This dissertation aims to demonstrate both types of applications that can rely on or exploit LN*, which are referred to as third-layer applications assuming that Bitcoin is the first and LN is the second layer.

Specifically, we first introduce a malicious use case of LN where a botmaster can control a botnet utilizing LN as the command and control (C&C) channel. In our design, we show that, unlike traditional or Bitcoin-based botnets, it is very hard to stop a botnet on LN due to LN's existing security and anonymity features. In our second work, we propose a secure and lightweight protocol to enable resource constrained IoT devices to use LN. With this protocol, IoT devices can send and receive LN payments by just involving in cryptographic signing operations. We implement this protocol by integrating it into LN's code and demonstrate that IoT devices can use it with minimal overhead to performance metrics. Finally, as a third work, we investigate fully offline Bitcoin payments which is of great need for communities that temporarily do not have access to the Internet. This usually happens when there is a natural disaster or a big scale power outage. We demonstrate that wireless mesh networks are a perfect venue to realize these offline payments without needing any extra infrastructure or protocol changes to LN or Bitcoin. We provide proof of concept implementations and ways to scale it to networks with much more people.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

In recent years, many cryptocurrencies started being used in various daily-life applications [CXS+18]. In particular, Bitcoin has been gaining tremendous popularity [Bro21] which was fueled by the revolutionary blockchain concept. Its market cap is now above 50% among all cryptocurrencies. Nevertheless, Bitcoin's transactions fees are still high and payment verification times are generally more than 10 minutes, which makes it unfeasible for real-time transactions. To address this issue, different schemes have been proposed [ZHZB20]. Among these, the most widely adopted one is the Lightning Network (LN). The idea is to utilize smart-contracts and avoid writing every transaction to the blockchain. Instead, the transactions are recorded off-chain until the accounts are reconciled. Specifically, once a channel is created between two peers, many off-chain transactions can be performed in both directions as long as there are enough funds. When many nodes come together, the off-chain payment channels turn into a network, referred to as payment channel network (PCN), such as LN. As of today, LN grew to more than 16k users in five years, making it a popular environment for instant Bitcoin transactions.

To briefly explain how LN off-chain mechanism works, we use an example case where Alice opens an LN channel to Bob with the purpose of sending him LN payments. When Alice wants to open a channel to Bob, she needs to construct a proper funding transaction first. This on-chain transaction determines the channel capacity which is the amount of funds that will be committed to the channel. Once Alice creates the transaction and receives Bob's signature, she broadcasts the funding transaction to the Bitcoin network. In Fig. 1.1, Alice opens a channel to Bob with a 5 Bitcoin capacity. Once funds are committed to the channel, she can send

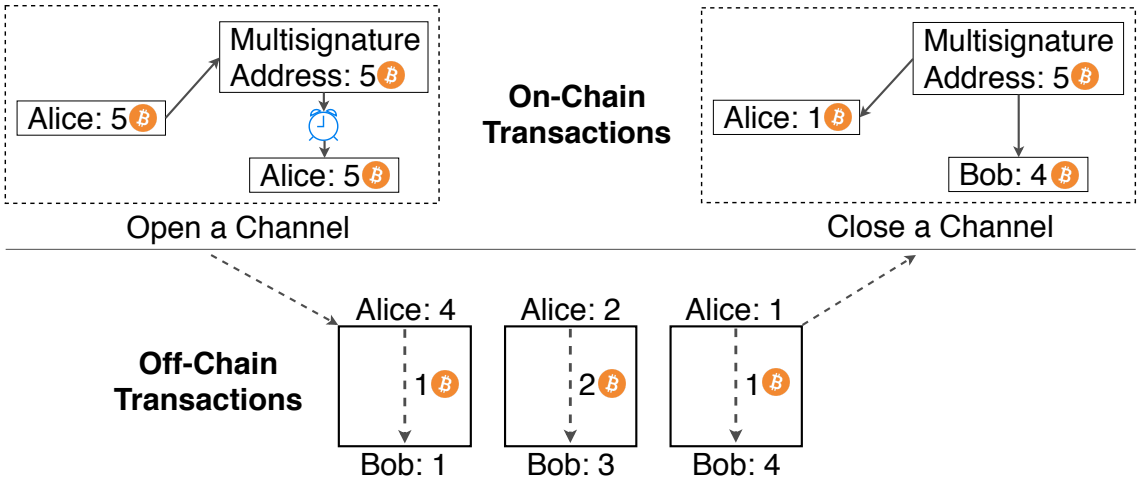

Figure 1.1: A depiction of the on-chain channel opening and closing transactions and off-chain payments in LN.

off-chain payments to Bob up to a total of 5 Bitcoins.

When Alice starts sending Bob off-chain payments, her and Bob's balances on the channel will change. In Fig. 1.1, Alice sends 3 different payments to Bob with amounts 1 Bitcoin, 2 Bitcoins, 1 Bitcoin respectively. Since these transactions are not on-chain (i.e., not mined by miners thus not included in the blocks), there has to be a different mechanism to keep track of each parties' balances in the channel. This is done by the *commitment transactions*. A commitment transaction is a type of Bitcoin transaction specifically designed for LN. A payment channel consists of states, changing with each payment. In each state, parties have different balances which are recorded onto their commitment transactions. She receives Bob's signature for each new state which enables her to broadcast her commitment transaction if required.

We also mention the security and privacy features of LN which we utilize in different parts of this dissertation. It can be listed as follows:

- **No publicly advertised activity:** The drawback of using a cryptocurrency based communication infrastructure is that all of the activities are publicly stored

in a persistent, transparent, append-only ledger. However, using the off-chain transaction mechanism, only the intermediary nodes in a multi-hop payment path know the transactions. The activities that are taking place in a multi-hop payment is locally stored by the nodes responsible for forwarding that payment.

- **Covert messaging:** LN was proposed to ease the problems occurring in the Bitcoin network. Hence, all of the actions taking place in the network is regarded as financial transactions. In that sense, twisting this idea into using the network for covertly forwarding commands will be indistinguishable from innocent, legitimate, and real financial transactions. Furthermore, even when the messages are attached to the payments, the communication is still covert as the intermediary nodes cannot tell that the payments include a message inside them.
- **Relationship anonymity:** LN utilizes onion routing [Lig22b] when forwarding the payments. This feature enables users to stay anonymous when transacting on LN. Intermediary nodes on a payment path cannot know the origin or the destination of the payment. This applies to any "curious" node in the network. Without colluding with other users, it is not possible to know who transacts with whom, which is known as the relationship anonymity.

1.1 Third Layer Bitcoin Applications

Emergence of LN opened new doors to many potential novel applications that can utilize its infrastructure. Indeed, LN's underlying network offers a perfectly covert communication medium to enable security and privacy by default. This creates opportunities for the sake of good and bad. *This dissertation aims to demonstrate both types of applications that can rely on or exploit LN*, which are referred to as third-layer applications assuming that Bitcoin is the first and LN is the second layer.

This dissertation tackles the challenges of building novel third layer LN applications on three practical use-cases.

Botnet: We first explored a malicious use case of LN where we show how LN can be used to control a botnet [KEC+20]. Numerous botnets have been proposed and deployed in the past [WSZ10, SMP18]. Regardless of their communication infrastructure being centralized or peer-to-peer, existing botnet C&C channels and servers have the challenge of remaining hidden and being resistant against legal authorities' actions. Hence, in Chapter 4, we first present *LNBot* [KEC+20], which is the first botnet that utilizes LN infrastructure for its communications between the botmaster and C&C servers with a two-layer hybrid architecture. This two-layer command and control mechanism not only enables scalability, but also minimizes the burden on each C&C server, which increases their anonymity. Second, we present *iLNBot* which stands for *improved version of LNBot* that has significantly reduced cost and latency overheads. However, the proposed LNBot and iLNBot still assume a centralized communication model between the botmaster and C&C servers. To further strengthen the anonymity, we show that a distributed version, namely *D-LNBot* can be created which forms itself over existing LN nodes.

IoT Micro-payments: Our second practical application is utilization of LN for enabling micro-payments (i.e., paying with your smart watch or vehicle) for resource constrained IoT devices. Although LN addresses many problems of Bitcoin, it still cannot be run on most of the IoT devices because of the computation, communication, and storage requirements [Kur21]. Therefore, a lightweight solution is needed. We are specifically focusing on Bitcoin's LN because LN is currently the most widely used cryptocurrency payment channel network. To this end, in Chapter 5, we propose a threshold cryptography-based protocol where an IoT device can perform LN operations through an *untrusted LN gateway* that hosts the full LN and Bitcoin

nodes [KMS+21]. With this integration, the IoT device can 1) open LN channels, 2) send LN payments, 3) receive LN payments, and 4) close LN channels. The LN gateway is incentivized to provide this payment service by charging service fees for IoT device's payments. By implementing our approach, we demonstrated that the proposed protocol 1) has negligible communication and computational delays thus enables timely payments; 2) is scalable for increasing number of IoT devices and payments; 3) can be run on networks with low bandwidth (data rate); and 4) associated energy consumption and monetary costs of using the protocol for IoT devices are negligible.

Offline Bitcoin Payments: Our third LN application is for enabling offline Bitcoin payments. Decentralized nature of Bitcoin offer opportunities to explore whether digital payments can be realized when there is no Internet connection. This is particularly crucial for the cases when there are natural disasters such as hurricanes or earthquakes causing power and Internet outages while people in communities still need to interact and make payments. Indeed, this was a particular issue after Hurricane Irma in 2017 when people in South Florida did not have Internet for weeks[1] while they still needed to make payments for gas, groceries, repairs and other basic needs. Thus, in Chapter 6, we propose using LN on top of community wireless mesh networks to enable sending/receiving offline LN payments between the members of the mesh network without needing any Internet connection. In this way, until users get back online, they can transact using their existing LN channels. As long as nodes can communicate with each other through wireless technologies such as WiFi or Bluetooth, they can perform offline LN payments.

[1] `https://mashable.com/article/hurricane-irma-power-outage-florida`

1.2 Our Contributions

Our contributions in this dissertation are as follows:

1. Contributions of Chapter 4: D-LNBot: A Scalable, Cost-Free and Covert Hybrid Botnet on Bitcoin's Lightning Network:

- We first propose LNBot which is a covert and hybrid botnet that uses Bitcoin's LN for its C&C communications. LN's strong anonymity features makes it very challenging to stop the botnet.
- Next, we propose iLNBot which is a significantly improved version of LNBot in terms of cost and time spent on sending the commands to the bots.
- We then propose D-LNBot which is a distributed version of LNBot that is superior in cost, delay and resiliency aspects. Specifically, D-LNBot forms itself over LN with minimum botmaster intervention, sends the commands to the bots for free, spends significantly less time to propagate the commands to all the bots.
- We present proof of concept implementations for all three versions and extensively analyze their performance metrics. Some implementation details can be found in our GitHub repositories at: `https://github.com/startimeahmet/D-LNBot` and `https://github.com/LightningNetworkBot/LNBot`.
- Finally, all these features of LNBot and D-LNBot make them botnets that need to be taken seriously therefore we provide a list of countermeasures that may help detect LNBot and D-LNBot activities and minimize damages from them.

2. Contributions of Chapter 5: LNGate^2: Secure Bidirectional IoT Micropayments using Bitcoin's Lightning Network and Threshold Cryptography:

- We propose LNGate^2, a secure and lightweight protocol that enables resource-constrained IoT devices to open/close LN channels, send/receive LN payments through an untrusted gateway node.
- We utilize (2,2)-threshold cryptography so that the IoT device does not have to get involved in Bitcoin or LN operations, but only in transaction signing and key generation. We propose to thresholdize LN's Bitcoin public/private keys and public/private keys of new channel states (i.e., commitment points) for a secure 2-party threshold LN node.
- We used game theory to analyze the security of the protocol and prove that it is secure against collusion attacks.
- We implemented LNGate^2 by changing LN's source code. LN's Bitcoin public and private keys were thresholdized. Our code is publicly available in our GitHub pages at: `https://github.com/startimeahmet/lightning` and `https://github.com/startimeahmet/LNGate2`.

3. Contributions of Chapter 6: LNMesh: Who Said You need Internet to send Bitcoin? Offline Lightning Network Payments using Community Wireless Mesh Networks:

- We propose LNMesh which enables using LN on top of community wireless mesh networks to enable sending/receiving offline LN payments between the members of the mesh network without needing any Internet connection.
- We conducted a feasibility study using 8 Raspberry Pi devices to realize the offline LN payments using Bluetooth Low Energy (BLE) and WiFi for communication.

- We then propose two channel assignment strategies based on minimum connected dominating set and universal spanning tree concepts to scale the concept to large-scale community mesh networks.
- We implemented a simulator in Python to assess the performance of the proposed channel assignment approaches and show that overall payment success rates up to 95% are achievable.
- Finally, the details of our proof of concept implementations and the full source code of our simulator can be found in our GitHub page at `https://github.com/startimeahmet/LNMesh`.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. We give the literature review of any related work in Chapter 2. It is followed by Chapter 3 to provide a comprehensive background for the proposed studies in the dissertation. Then, in Chapter 4, we propose D-LNBot which is a botnet utilizing LN for its covert communication. In Chapter 5, we propose LNGate^2 which is a secure and lightweight protocol enabling IoT devices to use the functions of LN. In Chapter 6, we propose LNMesh which enables making offline Bitcoin payments using wireless mesh networks. Finally, we give the concluding remarks and the future works in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we provide the related work of the studies presented in this dissertation.

2.1 Botnets

Botnets have been around for a long time and there have been even surveys classifying them [SSPS13, BCJ+09]. While early botnets used IRC, later botnets focused on P2P C&C channels for resiliency [GSN+07]. Our proposed LNBot and D-LNBot fall under covert botnets which became popular much later. As an example, Nagaraja et al. proposed Stegobot, a covert botnet using social media networks as a command and control channel [NHP+11]. Pantic et al. proposed a covert botnet command and control using Twitter [PH15]. Tsiatsikas et al. proposed SDP-based covert channel for botnet communication [TAK+15]. Calhoun et al. presented a MAC layer covert channel based on WiFi [CJCLB12]. A recent work by Wu et al. [WLH+21] presents a mobile covert botnet network. Another covert botnet design based on social networks was recently proposed by Wang et al. [WLC+22]. Tian et al. proposed DLchain [TGL+20], a covert channel utilizing the Bitcoin network.

Recently, there have been proposals on using the Bitcoin blockchain for botnet C&C communication [BAS+19]. For instance, Roffel et al. [RG14] came up with the idea of controlling a computer worm using the Bitcoin blockchain. Another work[Swe17] discusses how botnet resiliency can be enhanced using private blockchains. Pirozzi et al. [PP18] presented the use of blockchain as a command and control center for a new generation botnet. Kamenski et al. [KSWK21] also show a proof of concept to build a Bitcoin-based botnet. Similarly, ChainChannels [FAZ18] utilizes Bitcoin to disseminate C&C messages to the bots. These works are

different from our architecture as they suffer from the issues of high latency and public announcement of commands. ZombieCoin [AMLH15] proposes to use Bitcoin's transaction spreading mechanism as the C&C communication infrastructure. The authors later proposed ZombieCoin 2.0 [AMLH18] that employs subliminal channels to increase the anonymity of the botmaster. However, subliminal channels require a lot of resources to calculate required signatures which is computationally expensive and not practical to use on a large scale. A more recent work by Yin et al. [YCL+20] proposes CoinBot, a botnet that runs on cryptocurrency networks based on Bitcoin protocol such as Litecoin and Dash. Similar to other Bitcoin based botnets, CoinBot utilizes the `OP_RETURN` field in the transactions. However, this approach is still costly and command propagation is slow.

There are also Unblockable Chains [Zoh18], and Botract [AM22], which are Ethereum [Woo14] based botnet command and control infrastructures that suffer from anonymity issues since the commands are publicly recorded on the blockchain. Baden et al. [BFTFPS19] proposed a botnet C&C scheme utilizing Ethereum's Whisper messaging protocol. However, it is still possible to blacklist the topics used by the botmaster. Additionally, there is not a proof of concept implementation of the proposed approach yet, therefore it is unknown if the botnet can be successfully deployed or not.

The closest works to ours are the Franzoni et al. [FAD20] and DUSTBot [ZZZ+19]. Franzoni et al. propose to utilize the Bitcoin Testnet for controlling a botnet. Even though their C&C communication is encrypted, non-standard Bitcoin transactions used for the communication exposes the botnet activity. Once the botnet is detected, the messages coming from the botmaster can be prevented from spreading, consequently stopping the botnet activity. DUSTBot also partially uses the Bitcoin Testnet in its design. Authors propose to utilize the Testnet as the

upstream channel for sending the bot data back to the botmaster. However, botmaster's Bitcoin address is subject to blacklisting which blocks the communication of the botnet.

Finally, there are also botnets that utilize anonymity networks such as Tor [Bro10]. It is tempting for botmasters to use Tor and other anonymity networks due to their privacy and anonymity guarantees. However, there are numerous works in the literature showing ways to detect and stop botnets utilizing Tor. For example, Casenove et al. [CM14] showed that a botnet over Tor can be exposed due to the recognizable patterns in the C&C communication. Sanatinia et al. [SN15] proposed a Sybil mitigation technique to neutralize the bots in their proposed OnionBots which is a botnet utilizing Tor. Thus, as can be seen, anonymity networks are susceptible to other unique attacks associated with their inherent characteristics when used for botnet communication.

In contrast, our work is based on legitimate LN payments and does not require any additional computation to hide the commands. Also, these commands are not announced publicly. Moreover, LNBot offers a very unique advantage for its botmaster: C&C servers do not have any direct relation with the botmaster thanks to LN's anonymous multi-hop structure. Even more, D-LNBot removes the costs that were present in LNBot and enables running a free botnet on LN. As a result, LNBot and D-LNBot do not carry any mentioned disadvantages through their two-layer hybrid architecture and provide superior scalability and anonymity compared to others.

2.2 IoT Micro-payments

Hannon and Jin [HJ19] propose a protocol based on LN to give the IoT devices the ability to transact. They propose using two third parties that they named *IoT payment gateway* and *watchdog*. However, their approach has a fundamental flaw. They assumed that the IoT device can open payment channels to the gateway and this process is not explained. This is indeed the exact problem we are trying to solve. IoT devices do not have the computational resources to open and maintain LN payment channels. Therefore, their assumption is not feasible and our work is critical in this sense to fill this gap.

The authors of [RKG20] proposed IoTBnB which is a digital IoT marketplace that utilizes LN payments for data trading. In their approach, an *LN module* which hosts the Bitcoin and LN nodes is used to send users' payments. In contrast to our work, this approach is focusing on integrating LN into an existing IoT marketplace. Thus, the individual devices that are not part of such marketplaces are not considered. Additionally, the authors' LN framework relies on Bitcoin wallets held by the ecosystem itself which raises security and privacy concerns. In our approach, IoT devices do not share the ownership of their Bitcoins with a third party.

A work focusing on Ethereum micro-payments rather than Bitcoin was proposed by Pouraghily and Wolf [PW19]. It is a ticket-based verification protocol to enable low-end IoT devices to exchange money and data inside an IoT ecosystem. However, this approach has major problems: The joint account opened with a partner device raises security concerns as the details of it are not provided. Additionally, the approach was compared with μRaiden [Bra18] whose development stopped more than 4 years ago. In contrast to this work, we targeted Bitcoin's LN as it is actively being developed and dominating the market.

A recent work by Rebello et al. [RPBdAD22] proposed a hybrid PCN architecture for wireless resource constrained devices to be able to access and use PCNs. However, their solution does not work for devices that stay offline for long periods of time. Additionally, authors assume that resource-constrained devices can run light nodes and establish payment channels. This assumption cannot be generalized to all the IoT devices. Our approach does not require IoT devices to stay online except for the time they perform the LN operations. They also do not need to run light nodes.

Another recent work by Wang et al. [WZWX21] introduced HyperChannel which utilizes a group of Intel Software Guard Extensions (SGXs) that are run by selfish service nodes to execute the transactions. The approach includes an entity called *client emergency enclave*. This is an extra burden on the IoT devices since each device has to own an emergency enclave to protect themselves in case of service node shutdowns. In our approach, we only require IoT devices to perform signing.

Profentzas et al. [PAL20] proposed TinyEVM to enable IoT devices to perform micro-payments. The authors' method involves running a modified version of Ethereum virtual machine on the IoT devices. In contrast, in our approach, IoT devices only generate signatures which is not a resource-intensive operation. The work by Li et al. [LFX+20] focuses on designing a PCN-based smart contract for IoT data transactions. However, they do not seem to discuss the costs associated with their protocol and the routing performance of the protocol drops significantly when there are malicious nodes in the network. A slightly different work by Tapas et al. [TYL+20] proposed utilizing LN in the context of patch update delivery to the IoT devices where they claim rewards through LN. However, unlike our work, the IoT devices are assumed to be able to connect to the blockchain through light clients or third party full nodes on the network which requires a degree of trust.

There are also some implementation efforts for creating lighter versions of LN such as Neutrino [Lig23b] light client and Phoenix [ACI23] mobile wallet. The problem with these software is that they are not specifically designed for IoT devices thus most IoT devices cannot run them. Thus, we opt for a solution that covers a wider range of IoT devices and applications.

In addition to these works, we acknowledge that there are also many works that offer cash payments from IoT devices that do not include cryptocurrencies. These are typically bank-based, card-based and digital cash based payment options [WCZ20, BRV+22]. However, as mentioned before, our goal is not to compete with these solutions. Our work is just a reliable and convenient alternative for those who prefer cryptocurrency payments that rely on blockchains.

2.3 Offline Bitcoin Payments

While there has been a lot of work on the integration of Bitcoin with IoT devices [KMS+21, FCFL18], the concept of fully offline Bitcoin payments received little interest. The closest work to ours is from Myers [Mye19]. The author proposes a protocol called Lot49 that aims to incentivize message senders in a mesh network using LN-like payment channels. While the work looks interesting in the direction to possibly enable offline LN payments, it has too many requirements and assumptions that are unlikely to be feasible. For example, the protocol requires Schnorr signatures and `SIGHASH_NOINPUT` flag to be adopted by the Bitcoin community, changes to the Bitcoin scripts of LN, and many different types of nodes to be setup in the mesh. Additionally, there is no proof of concept implementation of the protocol. In contrast, our work does not need any modifications to the LN or Bitcoin protocols, and works without setting up so many different types of nodes.

It is also worth mentioning that, the idea of using mesh networks to enable offline Bitcoin and LN payments were mentioned at several news articles[1][2]. However, they do not offer any actual solution. There are also commercialized mesh networking solutions such as Locha Mesh[3] that might be used for sending Bitcoin offline. However, users need to purchase special nodes and the system is geared more towards communication or chat. Additionally, the open source development of the software seems to be dormant as of now.

A practical problem in this context is to temporarily accommodate offline LN nodes. While there was no scientific study, there have been discussions in the Bitcoin and Lightning community about how offline LN nodes can receive payments. Such payments are called asynchronous (async) payments. A thread on the Lightning-dev mailing list was started by Matt Corallo to discuss the possible solutions[4]. According to him, Point Time Locked Contracts[5] (PTLCs) which were proposed to replace HTLCs in LN, could be the best option. PTLCs use public key for locking and a corresponding signature for unlocking in contrast to HTLCs' hash and preimage combination. Since PTLCs are not yet implemented in LN; a partial solution, trampoline relays were proposed[6]. These relay nodes are managed by third parties and can temporarily hold the payments until the offline recipient node comes

[1] https://bitcoinmagazine.com/technical/making-bitcoin-unstoppable-part-one-mesh-nets

[2] https://bitcoinmagazine.com/technical/from-isp-to-p2p-how-mesh-networks-take-bitcoin-off-the-grid

[3] https://github.com/btcven/locha

[4] https://lists.linuxfoundation.org/pipermail/lightning-dev/2021-October/003307.html

[5] https://bitcoinops.org/en/topics/ptlc/

[6] https://github.com/ACINQ/eclair/pull/2435

back online. However, this approach does not address payment sending between two offline LN nodes. In our approach, we enable offline-to-offline payments utilizing wireless mesh networks.

There were some efforts on enabling offline but on-chain Bitcoin transactions as well which works based on the concept of online coin preloading. The first feasible idea was proposed by Dmitrienko et al. [DNY17] where authors proposed utilizing offline wallets leveraging secure hardware. Later, Takahashi et al. [TO20] slightly improved on this idea by removing the external trusted time-stamp server in the design. However, both approaches need wallets produced by trustworthy manufacturers which raises security concerns. Additionally, the approach requires changes to the Bitcoin protocol.

Researchers also explored other blockchains for realizing offline cryptocurrency payments. For example, Rawat et al. [RDS22] explored whether IOTA blockchain can be used to perform offline payments. They concluded that current IOTA blockchain cannot accommodate the desired offline payments without significant modifications to its protocol. A more concrete solution called *DelegaCoin* was proposed by Li et al. [LWZ+21] whose main idea is to utilize Trusted Execution Environments (TEEs) for secure offline delegation of coins without interacting with the blockchain. However, this approach requires additional entities to be set up (i.e., TEEs) which may not be practical for all users. In contrast to these works, we propose using LN on top of existing wireless mesh networks without needing to modify existing blockchain protocols or setting up new entities.

CHAPTER 3

BACKGROUND

This chapter provides background on LN, its underlying mechanisms, threshold cryptography and game theory as a preliminary to our studies presented in this dissertation.

3.1 Lightning Network Preliminaries

LN was introduced in 2015 in a draft technical whitepaper [PD16] and later was implemented and deployed onto Bitcoin Mainnet by Lightning Labs [Sub18]. It runs on top of the Bitcoin blockchain as a *second-layer* peer-to-peer distributed PCN and aims to address the scalability problem of Bitcoin. It enables opening secure payment channels among users to perform instant and cheap Bitcoin transfers through multi-hop routes within the network by utilizing Bitcoin's smart contract capability [ABC+18]. The number of users using LN has grown significantly since its creation. At the time of writing this chapter, LN incorporates 16,450 nodes and 74,625 channels which hold 5,438 BTC in total (worth around 160 million USD)[1].

Off-chain Concept: The main idea behind LN is to utilize the *off-chain* concept which enables near-instant Bitcoin transactions with negligible fees. This is accomplished through *bidirectional payment channels* which are created among two parties to exchange funds without committing the transactions to Bitcoin blockchain. The channel is opened by making a single on-chain transaction called the *funding transaction*. The funding transaction places the funds into a *2-of-2 multisignature* address which also determines the capacity of the channel. Whenever the parties want to send a payment, they basically shift corresponding portion of their channel balance to the other side of the channel. So, after a transaction takes place, the total ca-

[1] `https://1ml.com/`

pacity in the channel does not change but the directional capacities do. Therefore, the peers can make as many transactions as they want in any amount as long as the amount they want to send does not exceed the directional capacity. The example shown in Fig. 3.1 illustrates the concept in more detail.

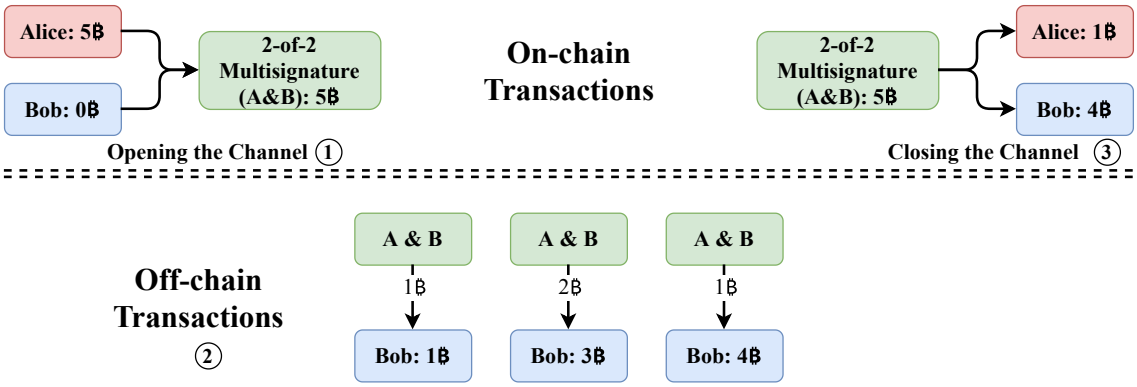

Figure 3.1: Off-chain mechanism of LN.

Per figure, ① Alice opens a channel to Bob by funding 5 Bitcoins into a 2-of-2 multi-signature address which is signed by both Alice and Bob. ② Using this channel, Alice can send payments to Bob simply by transferring the ownership of her share in the multi-signature address until her funds in the address are exhausted. Note that these transactions are off-chain meaning they are not written to the Bitcoin blockchain which is a unique feature of LN that is exploited in our botnet. Alice performs 3 transactions at different times with amounts of 1, 2 and 1 Bitcoin respectively. ③ Eventually, when the channel is closed, the remaining 1 Bitcoin in the multi-signature address is returned to Alice while the total transferred 4 Bitcoins are settled at Bob. Channel closing is another on-chain transaction that reports the final balances of Alice and Bob in the multi-signature address to the blockchain.

Multi-hop Payments: In LN, a node can send payments to another node even if it does not have a direct payment channel to that node. This is achieved by utilizing already established payment channels between other nodes and such a payment

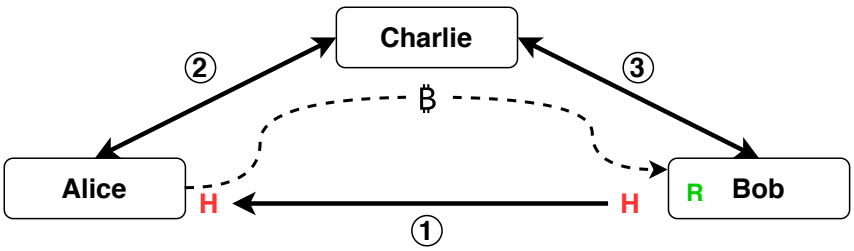

Figure 3.2: Illustration of a multi-hop payment. R is the pre-image generated by Bob, H is the hash of R. Alice creates an HTLC and includes H in it. Through this mechanism, Alice can securely route her payment to Bob over Charlie.

is called multi-hop payment since the payment is forwarded through a number of intermediary nodes (i.e., hops) until reaching the destination. This process is trustless meaning the sender does not need to trust the intermediary nodes along the route. Fig. 3.2 depicts a multi-hop payment. Alice wants to send a payment to Bob but does not have a direct channel with him. Instead, she has a channel with Charlie which has a channel with Bob, thus Alice can initiate a payment to Bob via Charlie. For this to work, ① Bob first sends an *invoice* to Alice which includes the hash (H) of a secret R (known as *pre-image*). ② Then, Alice prepares the payment through a contract called *Hash Time Locked Contract* (HTLC) [PD16] and includes H inside the payment. HTLCs are used to ensure that the payments can be securely routed over a number of hops. Now, Alice sends this HTLC to Charlie and waits for Charlie to disclose the R. That is the condition for Alice to release the money to Charlie. ③ In the same way, Charlie expects Bob to disclose R so that he can send it to Alice the claim the money. Finally, when Bob releases the R to Charlie, the payment will have successfully sent and HTLC will have fulfilled. Through this mechanism of LN, as long as there is a path from the source to the destination with enough liquidity, payments can be routed just like the Internet.

The rest of this section explains technical concepts about LN such as *funding transaction*, *commitment transaction*, *Hash Time Locked Contract (HTLC)*, *revoked*

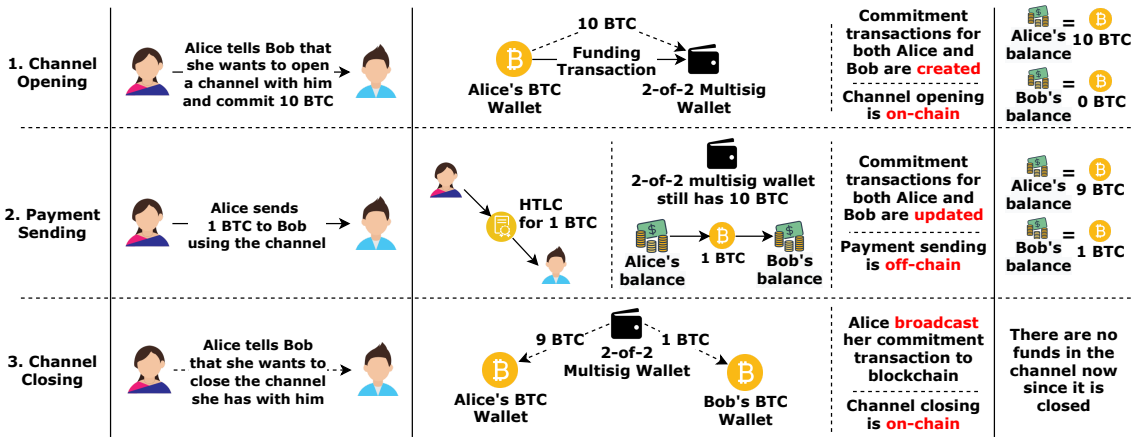

Figure 3.3: Life cycle of an LN channel.

state, and *Basis of Lightning Technology (BOLT)* which are essential to understand our protocol. The explanations are based on an example case where Alice opens an LN payment channel to Bob and wishes to transact with him.

Funding Transaction: When Alice would like to open a channel to Bob, this is done via an on-chain Bitcoin transaction called the *funding transaction*. Generally the channel is funded by the party initiating the process, but dual-funding where both channel parties commit funds to the channel is also possible. With the funding process, the capacity of the channel is determined as well. For example, if Alice funds the channel with 10 Bitcoins (BTC) as shown in Fig. 3.3 (1. Channel Opening), she can send payments to Bob until her 10 BTC in the channel are exhausted.

The reverse of the funding transaction is called the *closing transaction* which is used to close an LN channel (3. Channel Closing in Fig. 3.3). It is also an on-chain transaction which needs to be broadcast to the Bitcoin network.

Commitment Transaction: Once the channel is used for sending a payment, the balances of Alice and Bob in the channel will change. Since the LN payments are off-chain meaning they are not recorded on the Bitcoin blockchain, another type of Bitcoin transaction which will keep track of the channel balances is needed.

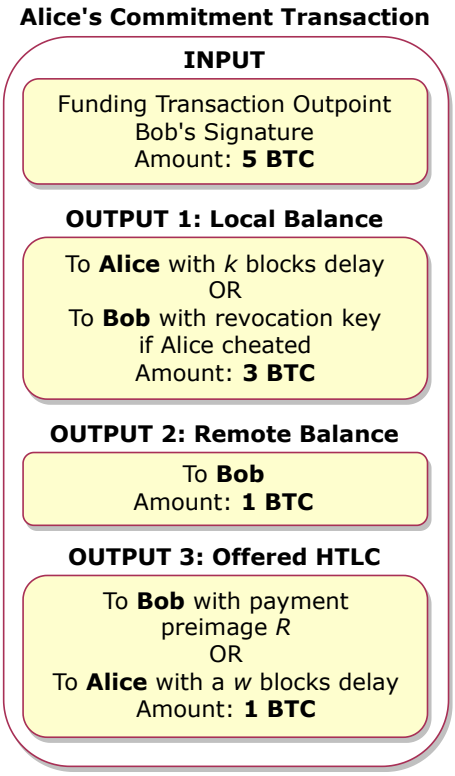

Figure 3.4: An illustration of a commitment transaction stored at Alice.

This is done by the *commitment* system of LN. A simple LN payment requires two separate commitment rounds: one when the payment is offered and one when it is fulfilled. Each commitment round requires both peers to sign a *commitment transaction*. Here, the commitment transactions are what actually hold the channel balance information and incoming/outgoing payments. They are specially crafted for LN and symmetrical for channel peers. An illustration of this is shown in Fig. 3.3 (2. Payment Sending) where Alice initiates a 1 BTC payment to Bob and their channel balances get updated.

Inputs to Alice's commitment transaction is the funding transaction outpoint and Bob's signature. Outpoint is the combination of the transaction output and its output index. A typical commitment transaction has three main outputs. For Alice's version, the first output is for her balance, the second output is for Bob's

and the third would be for a payment. These outputs are illustrated in Fig. 3.4.

Hash Time Locked Contract (HTLC): LN payments are constructed using a special type of transaction called Hash Time Locked Contract (HTLC). In this scheme, when Alice wants to send a 1 BTC payment to Bob as depicted in Fig. 3.3 (2. Payment Sending), she asks Bob to generate a secret called *preimage*. Bob hashes the preimage and sends the hash to Alice. Alice includes the hash in the HTLC and sends the HTLC to Bob. Upon receiving Alice's HTLC, Bob reveals the preimage to Alice to receive the 1 BTC payment. This acts as a proof that Bob is the intended recipient. If for some reason, Bob cannot reveal the preimage on time to claim the HTLC, Alice gets the 1 BTC back after some block height w. Output 3 in Fig. 3.4 illustrates this HTLC payment.

Revoked State: Alice's balance in her commitment transaction is conditional such that; if she closes the channel, she has to wait k number of blocks before she can redeem her funds on-chain. This is to protect Bob from a possible cheating attempt by Alice in which she uses an old (*revoked*) channel state to close the channel. Since Alice has a record of all the old channel states, it might be tempting for her to broadcast an old state in which she has more funds. But, if Bob sees that the channel was closed using an old state, he will punish Alice by taking all her funds in the channel. But to do that, he has to be online and take action before the broadcast transaction reaches the depth k on the blockchain. Note that, when Alice closes the channel, Bob can redeem his funds immediately unlike Alice.

Basis of Lightning Technology (BOLT): BOLT specifications describe LN's layer-2 protocol for secure off-chain Bitcoin payments. In order to implement our proposed protocol, we made modifications on BOLT #2 which is LN's peer protocol for channel management. BOLT #2 has three phases which are *channel establishment*, *normal operation* of the channel, and *channel closing*. Using this protocol,

LN nodes talk to each other for channel related operations. The full details of the protocol can be found at [Lig22a].

3.2 Invoice & Key Send Payments

LN payments are sent using *invoices* which are basically long strings consisting of characters and numbers[2]. They are encoded in a specific way to include all the information required to send the payment such as the destination public key, payment amount, expiration date, signature etc. The recipient of the payment prepares the invoice and sends it to the payer.

Key send in LN enables sending payments to a destination without needing to have an invoice first [Jag19]. It utilizes *Sphinx*[DG09] which is a compact and secure cryptographic packet format to anonymously route a message from a sender to a receiver. This is a very useful feature to have in LN because it introduces new use cases where payers can send spontaneous payments without contacting the payee first. In this mode, the sender generates the pre-image for the payment instead of the receiver and embeds the pre-image into the Sphinx packet within the outgoing HTLC. If an LN node accepts key send payments, then it only needs to advertise its public key to receive key send payments from other nodes. We utilize this feature to send payments from botmaster to C&C servers in LNBot, iLNBot and D-LNBot.

[2] `https://github.com/lightning/bolts/blob/master/11-payment-encoding.md`

3.3 Source Routing, Onion Routed Payments and Private Channels

With the availability of multi-hop payments, a routing mechanism is needed to select the best route for sending a payment to its destination. LN utilizes *source routing* which gives the sender full control over the route for the payment to follow within the network. Senders are able to specify: 1) the total number of hops on the path; 2) total cumulative fee they are willing to pay to send the payment; and 3) total time-lock period for the HTLC [Lig22b]. Moreover, all payments in LN are *onion-routed payments* meaning that HTLCs are securely and privately routed within the network. Additionally, by encoding payment routes within a Sphinx packet, the following security and privacy features are achieved: the nodes on a routing path do not know: 1) the source and the destination of the payment; 2) their exact position within the route; and 3) the total number of hops in the route. Consequently, these features prevent any node from easily censoring or analyzing the payments.

In LN, it is optional to announce the opened channels publicly. The channels that are not publicly announced to the network are called *private* channels. They are only known by the two peers of the channel. Other users cannot use the private channels to route payments over them. However, it is still possible to utilize these channels for routing payments with the help of *routing hints*[3]. Routing hints are used to let the sender know how to reach an LN node behind a private channel.

[3] `https://write.as/arshbot/everything-you-need-to-know-about-hop-hints-in-the-lightning-network`

3.4 Attaching Messages to the Payments

Recent developments in LN[4] made it easier to attach arbitrary data to the payments. A protocol to enable this functionality was developed at 2019[5]. Later, the protocol was incorporated to the Core Lightning through the noise plugin [Dec23]. We utilize the plugin for sending botmaster's commands to the C&C servers in iLNBot and D-LNBot. Messages are included in the payload inside the onion packets that are routed over the hops until reaching the recipient. Onion size is 1300 bytes, thus theoretically, messages of size up to 1300 bytes can be sent in a single payment. This is much higher than the size allowed by the `OP_RETURN` field in Bitcoin transactions which allows carrying only 80 bytes of data. The protocol carries the following information inside the onion payload: 1) key send pre-image, 2) message of variable size, 3) compressed signature and recovery id, and 4) timestamp. Intermediary nodes cannot the see payload inside the onion packet thus the messages can be anonymously sent to the recipients. Note that, different LN implementations might have a different version of this protocol or may not have it at all.

3.5 Threshold Cryptography

Threshold cryptography [Des94] deals with cryptographic operations where more than one party is involved. The idea of sharing a cryptographic secret among a number of parties was proposed by Shamir [Sha79]. In a threshold scheme, a secret is shared among n parties and a threshold t is defined such that, no group of $t-1$ can learn anything about the secret. Such setup is defined as (t, n)-threshold scheme. We utilize (2,2)-threshold cryptography in our proposed protocol for cooperative

[4] `https://github.com/lightning/bolts/pull/619`

[5] `https://github.com/joostjager/whatsat`

signing and key generation. Since Bitcoin is using Elliptic Curve Digital Signature Algorithm (ECDSA) for signing operations, in the next sections, we first present the ECDSA signature scheme. Then, we give the details of the 2-party ECDSA threshold key generation and signing that we employ in our protocol. They are rewritten from [Tea18] and based on Lindell's work [Lin17].

ECDSA Signature Scheme: ECDSA signature scheme takes an input message m and a private key x and produces a pair of integers (r, s) as output. The steps of the algorithm are as follows:

1. Hash $h = H(m)$ of the message is calculated. H is a hash function (i.e., SHA-256).
2. A secure random integer k between $[1, n-1]$ is generated.
3. A random point $(x, y) = k \cdot G$ is calculated. Then, $r = x \ (mod\ n)$ is computed.
4. Signature proof $s = k^{-1} \cdot (h + r \cdot x) \ (mod\ n)$ is calculated.
5. (r, s) is returned which is the ECDSA signature.

(2,2)-Threshold Key Generation: Let $\mathbb{G}$ be an elliptic curve of prime order q and a base point (generator) G. In our setting, the protocol is run between a server P_1 and a client P_2.

1. Initiation: Choosing a random x_1, the server computes $Q_1 = x_1 \cdot G$ and choosing a random x_2, the client computes $Q_2 = x_2 \cdot G$.
2. A variant of the ECDH key exchange (see [Lin17]) is run by the client and the server to generate the joint public key $Q = x_1 x_2 G$.
3. A Paillier key-pair is generated and $c_{key} = Enc_{pk}(x_1)$ is computed by the server where pk is the Paillier public key. Then, the server sends c_{key} and pk to the client.

4. The client receives a non-interactive zero-knowledge proof whose Paillier key is well-formed from the server. Proof in [GRSB18] as well as the parameters in [LN18] were utilized.

5. The client asks the server to prove that the encrypted value in c_{key} is the discrete log of Q_1 using a 2-round zero-knowledge protocol that is given at Section 6 of Lindell's work [Lin17].

6. The client asks the server to prove that $x_1 \in \mathbb{Z}_q$ where q is the order of the elliptic curve using a 2-round zero-knowledge protocol that is given at Appendix A of Lindell's paper [Lin17].

(2,2)-Threshold Signing: This 2-party threshold ECDSA algorithm produces the same signature (r, s) as the original ECDSA signature scheme given earlier.

1. Step 1 of the key generation protocol is repeated for ephemeral key-pairs by the server and the client. This outputs k_1, R_1 for the server and k_2, R_2 for the client.

2. Server and the client can extract the x-coordinate $r = r_x$ from the same R they generate by $R = k_1 \cdot R_2$ and $R = k_2 \cdot R_1$ respectively.

3. $c_1 = Enc_{pk}(k_2^{-1} \cdot H(m) + \rho q)$ and $c_2 = {c_{key}}^{x_2 \cdot r \cdot k_2^{-1}}$ are computed by the client where ρ is some random number. Then, $c_3 = c_1 \oplus c_2$ is computed and sent to the server by the client where $\oplus$ is the additive homomorphic operation of Paillier cryptosystem.

4. c_3 is decrypted by the server to get s' which is used to compute $s = s' \cdot k_1^{-1}$. Now, server outputs (r, s) as the ECDSA signature if (r, s) validates as a signature on $H(m)$.

3.6 Game Theory Background

In Chapter 5, the concept of *subgame perfect equilibrium* (SPE) [MR88] plays an important role to identify the solution. A (proper) subgame is a subset of the tree structure of the actual game. When tree form is used in a game, sequentiality can be applied rather than simultaneity. That is, in a two-player game, one player moves first and the other decides how to play after observing the first player's action. In this setting, it is assumed that the players are all sequentially rational, meaning that each player systematically and purposefully performs his/her best to achieve the objective. Most of the time, the objective for a player in an economic setting is to maximize profit. One should also note that profit maximization requires the cost minimization at the same time by its definition.

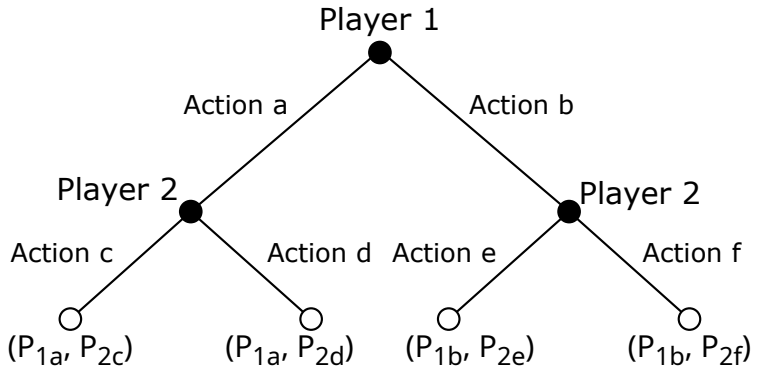

Figure 3.5: An example extensive form of a sequential game

To find the solution of a game with a similar setting (i.e., SPE), the concept called *backward induction* [Aum95] can be used. Applying the backward induction procedure requires investigating the game from the end to the start. Particularly, at each decision node, any action that results in a smaller payoff for the corresponding player is eliminated. Incorporating sequential rationality, a strategy profile is SPE if it specifies a *Nash equilibrium* [Mye78] in every subgame of the original game. A Nash equilibrium is the optimal solution which yields no incentive for any player in

the game to deviate once it is achieved. We show an example extensive form game in Fig. 3.5.

In this example, Player 1 and Player 2 are performing certain actions resulting in different payoffs at the end of the game.

CHAPTER 4

D-LNBOT: A SCALABLE, COST-FREE COVERT HYBRID BOTNET ON BITCOIN'S LIGHTNING NETWORK

While various covert botnets were proposed in the past, they still lack complete anonymization for their servers/botmasters or suffer from slow communication between the botmaster and the bots. In this chapter, we first propose a new generation hybrid botnet that covertly and efficiently communicates over Bitcoin Lightning Network (LN), called LNBot. Exploiting various anonymity features of LN, we show the feasibility of a scalable two-layer botnet which completely anonymizes the identity of the botmaster. In the first layer, the botmaster anonymously sends the commands to the command and control (C&C) servers through regular LN payments. Specifically, LNBot allows botmaster's commands to be sent in the form of surreptitious multi-hop LN payments, where the commands are either encoded with the payments or attached to the payments to provide covert communications. In the second layer, C&C servers further relay those commands to the bots in their mini-botnets to launch any type of attacks to victim machines. We further improve on this design by introducing D-LNBot; a distributed version of LNBot that generates its C&C servers by infecting users on the Internet and forms the C&C connections by opening channels to the existing nodes on LN. In contrary to the LNBot, the whole botnet formation phase is distributed and the botmaster is never involved in the process. By utilizing Bitcoin's Testnet and the new message attachment feature of LN, we show that D-LNBot can be run for free and commands are propagated faster to all the C&C servers compared to LNBot. We presented proof-of-concept implementations for both LNBot and D-LNBot on the actual LN and extensively analyzed their delay and cost performance. Finally, we also provide and discuss

a list of potential countermeasures to detect LNBot and D-LNBot activities and minimize their impacts.

4.1 LNBot Architecture

In this section, we describe the overall architecture of LNBot with its elements.

4.1.1 Overview

The overall architecture is shown in Fig. 4.1. As shown, the LN is used to maintain the C&C servers and their communication with the botmaster. Each C&C server runs a separate mini-botnet. Receiving the commands from the botmaster, the C&C servers relay the commands to the bots in their possession to launch attacks. The details of setting up the C&C servers are explained next.

4.1.2 Setting up the C&C Servers

The botmaster sets up the necessary number of C&C servers s/he would like to deploy. Depending on the objectives, the number of these servers and the number of bots they will control can be adjusted without any scalability concern. In Section 4.3, we explain how we set up real C&C servers running on LN on the real Bitcoin network.

Each C&C server is deployed as a *private* LN node which means that they do not advertise their channels within the LN. This helps C&C servers to stay anonymous in the network. Each C&C server opens private channels to at least k different random public LN nodes. The number k may be tuned depending on the size and topology of LN when LNBot will have deployed in the future. To open the channels, C&C servers need some Bitcoin in their Bitcoin wallets. This Bitcoin is provided to the

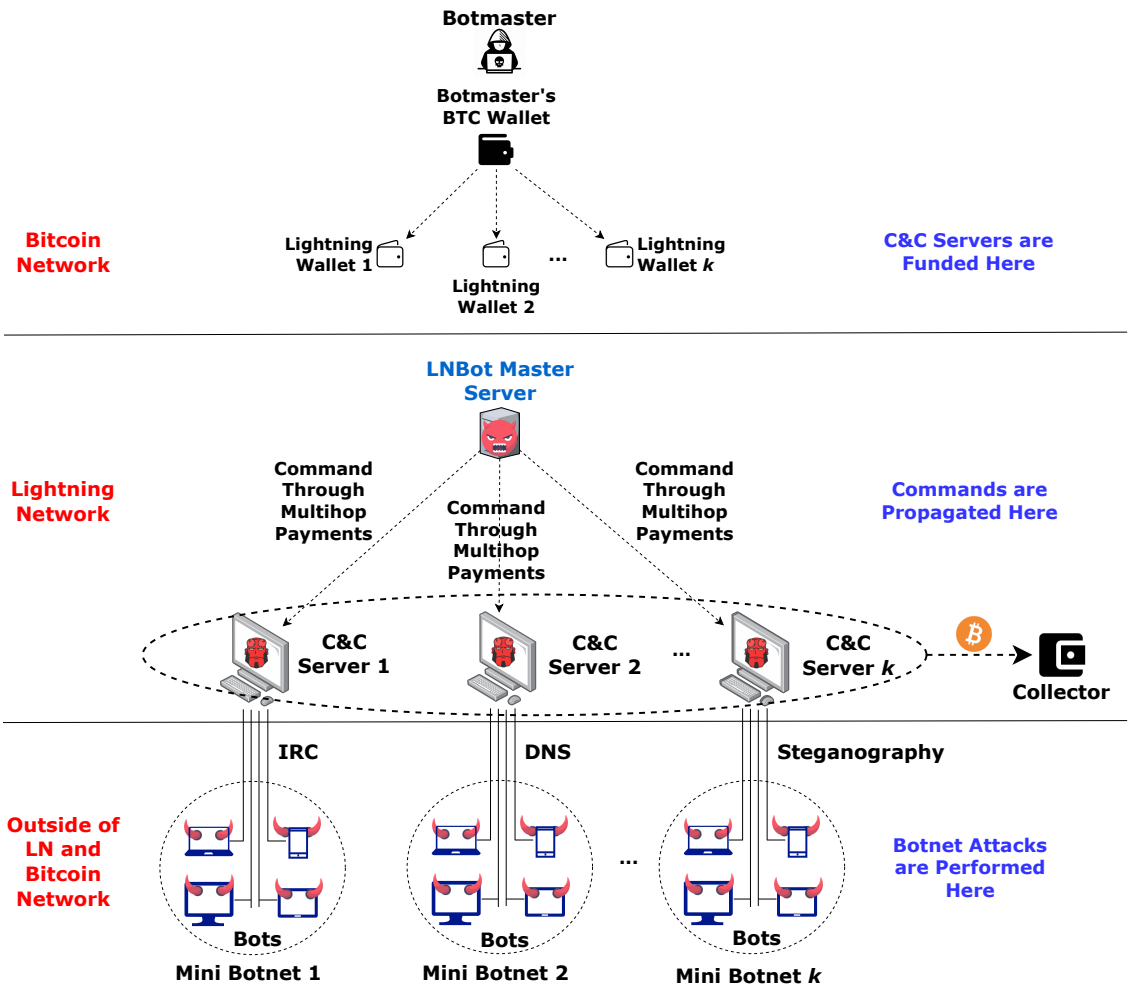

Figure 4.1: Overview of LNBot architecture.

C&C servers by the botmaster before deploying them. Since C&C servers' channels are unannounced (i.e., private) in the network, botmaster uses *routing hints* to be able to route his/her payments to the C&C servers. In this case, since the C&C servers are set up by the botmaster, s/he knows the necessary information to create the routing hints.

4.1.3 Formation of Mini-botnets

After the C&C servers are set up, we need bots to establish connections to the C&C servers. An infected machine (bot) connects to one of the C&C servers available. The details of bot recruitment and any malware implementation issues are beyond the objectives of this work. It is up to the botmaster to decide which type of infrastructure the C&C servers will use to control the bots in their possession. This flexibility is enabled by our proposed two-layer hybrid architecture of the LNBot. The reason for giving this flexibility is to enable scalability of LNBot through any type of mini-botnets without bothering for the compromise of any C&C servers. As it will be shown in Section 4.5, even if some of the C&C servers are compromised, this neither reveals the other C&C servers nor the botmaster.

4.1.4 Forming LNBot

Now that the C&C servers are set up and mini-botnets are formed, the next step is to form the infrastructure to control these C&C servers covertly with minimal chances of getting detected. This is where LN comes into play. Botmaster has the LN public keys of all the C&C servers. Using an LN node called *LNBot master server*, the botmaster initiates the commands to all the C&C servers through LN payments. Similar to the C&C servers, LNBot master server is also a private LN node and the botmaster has flexibility on the setup of this node and may change it regularly. Without using any other custom infrastructure, the botmaster is able to control the C&C servers through LN, consequently controlling all the bots in the botnet.

4.1.5 Command Propagation in LNBot

Once the LNBot is formed, the next step is to ensure communication from the botmaster to the C&C servers. We utilize a *one-to-many* architecture where the botmaster sends the commands to each C&C server separately. Commands can be sent in two different ways: 1) Encoding the characters in the command to individual LN payments; 2) Attaching the whole command to a single LN payment. The first method is utilized in LNBot and the second one is utilized in *iLNBot*. These two methods are explained separately at the sections below.

Encoding/Decoding Schemes

An important feature of LNBot is its ability to encode botmaster's commands into a series of LN payments that can be decoded by the C&C servers. We used ASCII encoding and Huffman coding [Huf06] for the purpose of determining the most efficient way of sending commands to the C&C servers in terms of Bitcoin cost and time spent. American Standard Code for Information Interchange (ASCII) is a character encoding standard that represents English characters as numbers, assigned from 0 to 127. Huffman coding on the other hand is one of the optimal options when the data needs to be losslessly compressed. We used Quaternary numeral system to generate the codebook as will be shown in Section 4.4.

For both methods, botmaster uses the Algorithm 1 to send the commands. S/he first checks if the respective C&C server is online or not (LN nodes have to be online in order to send and receive payments) before sending any payment. If the C&C server is not online, command sending is scheduled for a later time. If online, the botmaster sends 5 satoshi as the special starting payment of a command, then the actual characters of the command one by one. Lastly, the botmaster sends 6 satoshi as the special ending payment to finish sending the command. Note that the

Algorithm 1: Send Command

```
1  define k;
2  initialize command;
3  int counter = 0;                                   /* retry count */
4  bool isOnline = checkIfC&CServerIsOnline();
5  if isOnline then
6      bool result = send(5 satoshi);
7      if result==success then
8          counter = 0;
9          for character in command do
10             bool result = send(character);
11             if result==success then continue;
12             else if result==fail and counter < k then
13                 retry sending character;
14                 counter++;
15             else reschedule(command, date, time);
16         end
17         counter = 0;
18         bool result = send(6 satoshi);
19         if result==success then
20             Command has been successfully sent!;
21         else if result==fail and counter < k then
22             retry sending 6 satoshi;
23             counter++;
24         else reschedule(command, date, time);
25     else if result==fail and counter < k then
26         retry sending 5 satoshi;
27         counter++;
28     else reschedule(command, date, time);
29 else
30     reschedule(command, date, time);
31 end
```

selection of 5 satoshi and 6 satoshi in this algorithm depends on the chosen encoding and could be changed based on the needs. If any of these separate payments fail, it is retried. If any of the payments fail for more than k times in a row, command sending to the respective C&C server is canceled and scheduled for a later time.

Noise Plugin Method

When LNBot was introduced, attaching custom data to LN payments was a work in progress by the LN developers. As explained in Section 3.4, noise plugin [Dec23] gives one the opportunity to embed messages in LN payments. If we utilize this plugin in LNBot for sending botmaster's commands, the command sending will be much faster and cheaper. Specifically, the delay of sending a command to a C&C server will be reduced to a delay of sending a single payment. In the same way, the cost of sending a command will now be the cost of sending a single payment. Thus, we propose *iLNBot* which utilizes the noise plugin as its command sending mechanism instead of the encoding method presented earlier.

4.1.6 Reimbursing the Botmaster

Another important feature of LNBot is the ability of its botmaster to get the funds back from the C&C servers. Depending on botmaster's command propagation activity, C&C servers' channels will fill up with the funds received from the botmaster. Therefore, in our design, C&C servers are programmed to send the funds in their channels to an LN node called *collector* when their channels fill up completely. Collector is a private LN node set up by the botmaster. Its LN public key is stored in the C&C servers and thus they can send the funds to collector through LN. In addition to collector's LN public key, C&C servers are also supplied with routing hints to be able to successfully route their payments to the collector. In this way, the funds accumulate at the collector. The botmaster can collect these funds by closing collector's respective LN channels and sending the settled Bitcoins to his/her own Bitcoin wallet through on-chain transfers.

4.2 D-LNBot Architecture

The LNBot we presented in the previous section relies on a centralized model where the botmaster communicates with each C&C server separately. Additionally, in LNBot, C&C servers are manually set up by the botmaster which incurs several operating costs on the botmaster that grows with each installed C&C server. Specifically, botmaster needs to create machines to run the C&C servers and the LN nodes on them as well as fund each of these LN nodes which costs time and money. On top of that, LNBot's centralized design might tamper the anonymity of the botmaster and the C&C servers. Therefore, in this section, we present a distributed version of LNBot, namely D-LNBot, where the payments sent by the botmaster are reduced to a single transaction (i.e., to a specific C&C server) regardless of the size of the botnet. On top of the distributed design, we propose to utilize the Bitcoin Testnet instead of the Mainnet which removes the associated costs of running the botnet.

4.2.1 Overview

In this section, we describe the overall architecture of D-LNBot with its elements. LN is used for the communication between the botmaster and the C&C servers also among the C&C servers themselves. Different than LNBot, in D-LNBot, botmaster does not send the commands to each C&C server individually. Rather, the commands are just sent to one of the C&C servers which relays them to all other C&Cs distributively as will be explained in the subsequent sections. We illustrated this difference in command sending between the LNBot and D-LNBot in Fig. 4.2. We propose to utilize the noise plugin for sending the commands. Similar to the LNBot, each C&C server controls a mini-botnet which can utilize any existing known C&C infrastructure in the literature.

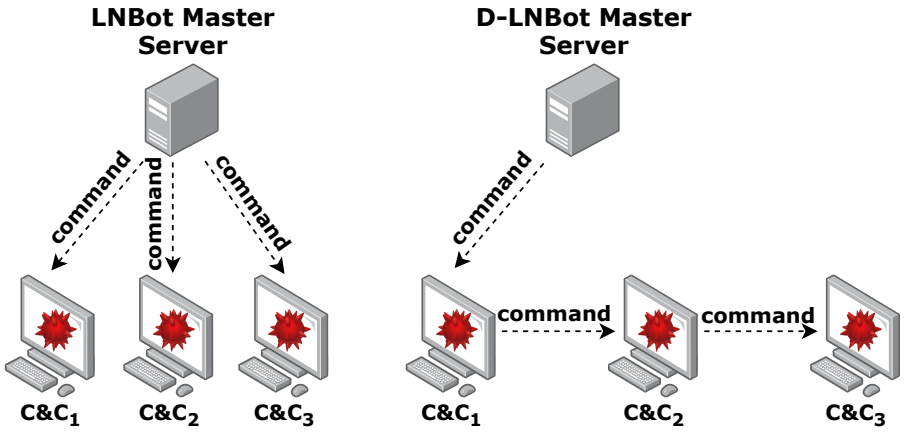

Figure 4.2: Comparison of the command propagation between LNBot and D-LNBot.

4.2.2 Creation of the C&C Servers

The way that the C&C servers are created differs from the LNBot. Botmaster infects existing machines on the Internet to turn them into C&C servers. Thus, s/he deploys a malware into the wild for this purpose. Deployment details of this malware is out of the scope of this work. However for later reference, we will call it the *malware_1*. Once a machine is infected with *malware_1*, there are two possibilities: 1) With probability p, the machine becomes a C&C server or 2) with probability *1-p*, nothing happens and the malware deletes itself. Ultimately, the probability p should be less than 0.5 to ensure that C&C servers are generated occasionally. The reason for not turning every infected machine into a C&C server is to limit the number of C&C servers that will be ever created. If botmaster lets every infected machine to turn into a C&C server, that will increase the number of public LN nodes in LN on Bitcoin's Testnet dramatically. As of May 2023, LN on Bitcoin's Testnet have around 2,400 public nodes[1]. Considering how fast a malware can infect new machines, we propose that it is best to limit the number of C&C servers that can be generated with the malware. There might be several ways to

[1] `https://1ml.com/testnet/`

do this and we leave its design to the botmaster. After this process, if the machine turns into a C&C server, it will download an LN software first. We propose to go for an LN light client instead of a full LN node since downloading a full LN node may alert the user of the infected machine as the download size is around 480 GB at the time of writing this chapter. Light LN clients such as Neutrino [Lig23b] occupy a very small space (around 1 MB) on the file system and have higher chances of going unnoticed by the users. After the LN node is created and initiated, it needs to be funded with some Testnet Bitcoins. For this, we suggest to use a pre-funded Testnet Bitcoin wallet where the C&C server can fetch the Bitcoins from. Generally, Testnet Bitcoins are obtained from Testnet faucets[2]. The procedure of pre-funding a wallet can be tailored by the botmaster to leave as little on-chain trace as possible. After the C&C server obtains some Testnet Bitcoins, the next step is the recruitment of the bots to work under the C&C server (i.e., formation of the mini-botnet).

4.2.3 Formation of the Mini-botnets

Similar to the LNBot, the botmaster uses a specific malware for this purpose. Let us call it *malware_2*. The machines that are infected with *malware_2* become bots and connect to the available C&C servers. When enough number of machines are infected with *malware_2*, they form a mini-botnet which are controlled by their respective C&C server. Here, the command and control infrastructure to control the bots can be different for each mini-botnet. There are many available C&C types such as DNS, social media, IRC, blockchain etc. [VZF17]. Botmaster should include as many of these C&C channel options as possible into the *malware_2* so that the C&C servers can randomly utilize one of them. These randomization is required since in D-LNBot, the C&C servers are not set up by the botmaster. Otherwise,

[2]e.g., `https://coinfaucet.eu/en/btc-testnet/`

botmaster would have to create variations of *malware_2* which utilize different C&C channels. However, this creates additional management overhead for the botmaster therefore we opt to create only one malware for this purpose that includes different C&C channel options where C&C servers can randomly choose from. This flexibility is a design choice which enables creating a *hybrid botnet*. So, ultimately botmaster deploys two malware into the wild, namely *malware_1* and *malware_2*, to create the C&C servers then to create bots to work under them. Next step is to create the connections between the C&C servers to form the D-LNBot.

4.2.4 Forming D-LNBot using Innocent Nodes

An important challenge in D-LNBot is to piece together the D-LNBot from individual C&C servers without the botmaster intervening in the process. We propose forming the communication among the C&C servers by exploiting the connections that can be created with existing nodes in LN. We will call them *innocent nodes* and basically these are LN nodes that are publicly reachable by other nodes. Current LN implementation lets users explore the network topology. For example, typing `lncli describegraph` (`lncli` is *lnd's* [Lig23a] command line interface) returns the network topology information in JSON format that can be analyzed to get information about the nodes reachable from our node. The information that can be collected include their public keys, IP addresses, channels, and channels' capacities.

The steps of forming the D-LNBot is shown in Algorithm 2 as well as illustrated in Fig. 4.3. First step is to identify some innocent nodes. When a new C&C server, say C&C$_n$, is created, it will establish channels with some of these innocent nodes following a specific policy. To do that, first, C&C$_n$ queries the LN and finds the h most connected nodes in the network. Next, among these h nodes, C&C$_n$ ran-

domly selects one of them and opens a channel to it with a capacity K_1 (line 7 in Algorithm 2). As for choosing the value of K_1, it should satisfy the following condition: $f(K_1) = \xi$, where $f()$ is a general function and ξ is an unique value (line 3-6). Then, C&C$_n$ scans the LN to discover the existing C&C servers (line 8). This is easy to do by querying the LN and checking the capacity of each channel to see if it is satisfying the rule given earlier. As soon as C&C$_n$ finds a channel satisfying the rule, it will register the node that opened this channel as a C&C server to its local database (line 16). Here, even though discovering the existing C&C servers is the goal, it is not desirable to reveal the presence of all existing C&C servers at once. Therefore, older C&C servers should hide their existence and the newly joined C&C servers should be able to discover only a small number of C&C servers at any given time. This is possible by older C&C servers closing their open channels with the innocent nodes after getting discovered by m new C&C servers (line 31). For example, if m is set to 2, C&C$_1$ will close its open channel with the innocent node after getting discovered by the C&C$_3$. C&C$_2$ will do the same after C&C$_4$ joins the network. In this way, there will always be m C&C servers only that are discoverable by a newcomer at any time. This number can be adjusted based on the needs and we will refer to it as the *number of active C&C servers*. But, how can the existing C&C servers know that a new C&C server joined the network? This is possible by existing C&C servers monitoring each new channel creation on LN and checking if the channel capacity satisfies the given rule. New channels are already announced to the network automatically so one would only need to implement a helper function to detect the channels that were created by the newly joined C&C servers (line 10). Following this mechanism, each C&C server will be aware of m older C&Cs and m newly joined C&Cs (except for the first m C&C servers).

Next, assume that a new C&C server, C&C$_{n+1}$, joins the network. Similarly, it will establish a channel with an innocent node with a capacity K_2 where $f(K_2) = \xi$ and query the LN to find the older C&Cs. It will find out that, C&C$_n$ has a channel with an innocent node that satisfies the condition $f(K_1) = \xi$. Thus, C&C$_{n+1}$ will add C&C$_n$ as a C&C server to its local database (line 16). In the mean time, C&C$_n$ will observe that C&C$_{n+1}$ has joined LN since it opened a channel satisfying the rule $f(K_2) = \xi$ and will add C&C$_{n+1}$ to its local database (line 28). This procedure goes on as new C&C servers join the network. Here, the function $f()$ and value ξ can be hard-coded in the *malware_1*. An example function for $f()$ could be $sin()$. One possible issue that might arise is that when new C&C servers join the network, the h most connected nodes in LN might change. This might cause the newly joined C&C servers to not be able to discover the existing C&C servers and vice versa. To remedy this issue, the C&C servers that could not discover at least two other C&C servers repeat the process of opening a channel to an innocent node (line 20). This will make sure that all C&C servers are aware of at least two other C&C servers. Essentially, this process forms an *overlay network* on top of LN to control a botnet.

4.2.5 Command Propagation in D-LNBot

Unlike LNBot, D-LNBot utilizes a *logical topology* created during the setup for command propagation. To initiate the command propagation, the botmaster sends the command from a node called *D-LNBot master server* to a particular C&C server which sends it to the C&C servers it recorded in its local database. We will call these recorded C&Cs, *neighboring C&C servers*. Every C&C server does the same so the command is propagated among the C&C servers in a P2P fashion. We illustrate two logical D-LNBot topologies in Fig. 4.4 when the number of active C&C servers

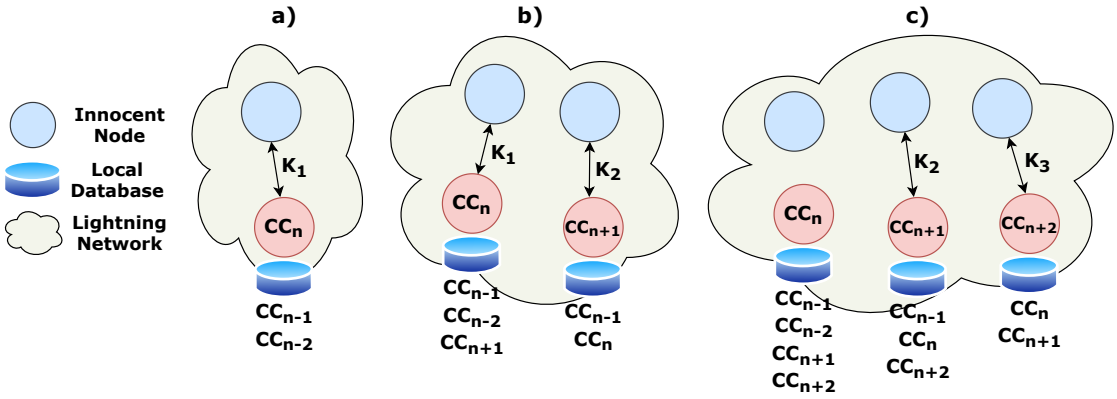

Figure 4.3: An illustration of how D-LNBot is formed when number of active C&C servers is 2 (i.e., $m = 2$). a) C&C$_n$ joins the network and opens a channel to an innocent node with a capacity K_1. b) C&C$_{n+1}$ joins the network and opens a channel to an innocent node with a capacity K_2. c) C&C$_{n+2}$ joins the network and opens a channel to an innocent node with a capacity K_3 which results in C&C$_n$ closing its channel with the innocent node. The process of C&C servers registering each other as neighboring servers into their local databases are also illustrated.

is two and three. The bidirectional links show the command propagation between the respective C&C servers. Note that, the C&C servers are not connected to each other with direct channels.

For the $m = 2$ case in Fig. 4.4, the propagation starts when C&C$_1$ receives a command from the botmaster. C&C$_1$ immediately relays the command to its neighbors C&C$_2$ and C&C$_3$ by sending them a single payment which includes the command. Then, when C&C$_2$ receives the command, it also immediately forwards the command to its neighbors C&C$_1$, C&C$_3$, and C&C$_4$. C&C$_3$ does the same when it receives the command from C&C$_1$ or C&C$_2$. Here, each C&C server sends and receives the command multiple times but this do not result in multiple executions of the same command. Because, each C&C server knows its neighbors and upon receiving a command from one of its neighbors, the C&C server can ignore the command if it was received from another neighbor before. This requires being able to authenticate the sender of the commands which the C&C servers can do. As mentioned in Section 3.4, the noise plugin messages include a *signature*. Using the

Algorithm 2: Form D-LNBot

```
define h, m, f(), ξ;
global K = [];                                   /* an empty list */
for i ← 1 to n do
    /* calculate n valid channel capacities for the policy based
       on f() and ξ                                                */
    solve f(K_i) = ξ for K_i;
    K.append(K_i);
end
global node = open_channel_to_innocent();
check_for_existing_C&Cs();
global counter = 0;                              /* newcomer C&C count */
monitor_new_channels();
Function check_for_existing_C&Cs(void):
    LN_topology = query_LN();
    count = 0;                                   /* existing C&C count */
    for channel in LN_topology do
        if f(channel.capacity) == ξ then
            register channel.getNode();
            count++;
        end
    end
    if count < 2 then open_channel_to_innocent();
end
Async Function monitor_new_channels(void):
    /* will catch each new channel                                 */
    return is_C&C(channel);
end
Function is_C&C(channel):
    if f(channel.capacity) == ξ then
        if counter < m then
            register channel.getNode();
            counter++;
        else
            close_channel(node);
        end
    end
end
Function open_channel_to_innocent(void):
    innocent_nodes = [];                         /* an empty list */
    LN_topology = query_LN();
    innocent_nodes = get_h_most_connected(LN_topology, h);
    node = randomly_select_one(innocent_nodes);
    K_1 = randomly_select_one(K);
    open_channel(node, K_1);
    return node;
end
```

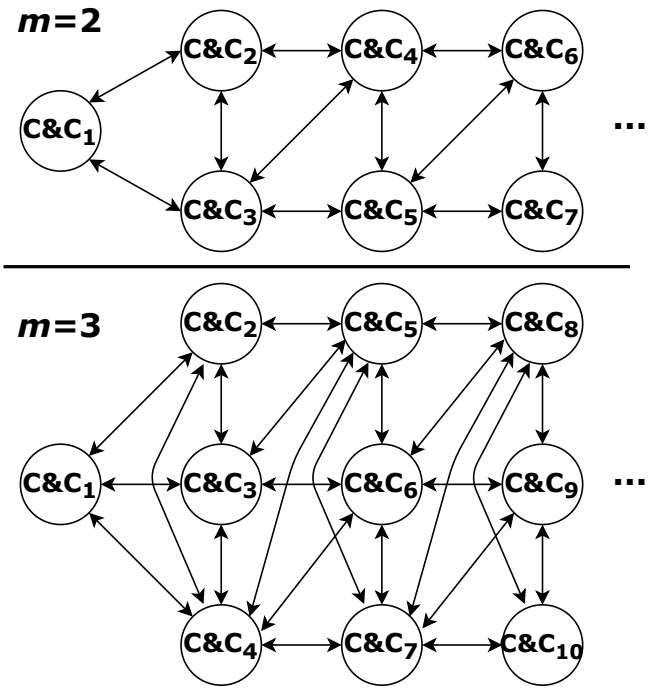

Figure 4.4: An illustration of the logical topology and command propagation of two sample D-LNBots when the number of active C&C servers is two and three respectively (i.e., $m = 2$ and $m = 3$). In this example, botmaster initiates the command sending from C&C$_1$.

signature, the LN public key of the sender can be recovered. This means that, a C&C server can compare this public key with the ones it has in its local database to make sure the command is coming from a legitimate C&C server. Additionally, the redundancy on sending and receiving the commands helps C&C servers keep their channels balanced. For the command propagation of $m = 3$ case, please refer to Fig. 4.4.

The redundancy on command propagation also helps detecting any false positives during the D-LNBot formation phase. Let us imagine a scenario where an honest LN node opens a channel with a capacity satisfying the policy used by the C&C servers. This will be automatically detected by the active C&C servers (there are only m of them) who were actively querying the LN to detect the newly joined C&C servers. The active C&C servers will register this honest LN node to their

local database. Now, let us imagine that the botmaster issues a command which starts getting propagated among the C&C servers which will eventually reach the C&Cs that registered the honest node as a neighbor C&C. At that point, the honest node will receive a number of noise messages from these C&Cs but will not propagate anything back. In fact, it might not even receive the messages successfully because of the custom TLV (type-length-value) used by the noise messages. Regardless, when the C&Cs do not get the command back from the honest node, they can understand it was not actually a C&C server thus, mark it as a false positive and delete it from their local database.

One advantage of D-LNBot is that, the botmaster can initiate the command propagation from any one of the C&C servers because the initial C&C server will replicate the command to its neighboring C&C servers. In other words, the botmaster does not need to initiate the command propagation from $C\&C_1$ necessarily, and can use the $C\&C_5$ instead for example. Botmaster is able to do this because s/he is aware of all the C&C servers as s/he is constantly watching the network and knows the policy the C&C servers use to open channels to the innocent nodes. Being able to freely choose the first C&C server to initialize the command propagation also provides better anonymity for the botmaster.

4.3 Proof-of-Concept Implementation

In this section, we demonstrate that implementations of the proposed LNBot and D-LNBot are feasible by presenting a proof-of-concept for each. Full LN nodes interact with the Bitcoin network in order to run the layer-2 protocols. There are two Bitcoin networks: *Bitcoin Mainnet* and *Bitcoin Testnet*. As the names suggest, Bitcoin *Mainnet* hosts Bitcoin transfers with a real monetary value. On

the contrary, in Bitcoin *Testnet*, Bitcoins do not have a monetary value. They are only used for testing and development purposes. Nonetheless, they both provide the same infrastructure and LNBot and D-LNBot can run on both networks. The only difference between the Testnet and Mainnet networks for LN is the number of nodes and the channels they have. LN on Bitcoin Testnet have about 60% less nodes, and 80% less channels compared to LN on Bitcoin Mainnet.

LNBot: We used *lnd* (version 0.9.0-beta) from Lightning Labs [Lig23a] for the full LN nodes. We used *Bitcoin Testnet* for our proof-of-concept development. We created 100 C&C servers and assessed certain performance characteristics for command propagation. We created a GitHub page explaining the steps to set up the C&C servers[3]. The steps include installation of *lnd* & *bitcoind*, configuring *lnd* and *bitcoind*, and extra configurations to hide the servers in the network by utilizing private channels. Nevertheless, to confirm that the channel opening costs and routing fees are exactly the same in both Bitcoin *Mainnet* and *Testnet*, we also created 2 nodes on Bitcoin *Mainnet*. We funded one of the nodes with 0.01 Bitcoin, created channels and sent payments to the other node. We observed that the costs and fees are exactly matching to that of Bitcoin *Testnet*.

lnd has a feature called *autopilot* which opens channels in an automated manner based on certain initial parameters set in advance[4]. Our C&C servers on Bitcoin *Testnet* employ this functionality of *lnd* to open channels on LN. Using *autopilot*, we opened 3 channels per server. Note that this number of channels is picked based on our experimentation on Bitcoin *Testnet* on the success of payments. We wanted to prevent any failures in payments by tuning this parameter. As mentioned, these 3 channels are all private, created with `--private` argument, which do not broadcast

[3] `https://github.com/LightningNetworkBot/LNBot`

[4] `https://github.com/lightningnetwork/lnd/blob/master/sample-lnd.conf`

themselves to the network. A private channel in LN is only known by the two peers of the channel.

lnd has an API for communicating with a local *lnd* instance through gRPC[5]. Using the API, we wrote a client that communicated with *lnd* in Python. Particularly, we wrote two Python scripts, one running on the C&C servers and the other on the botmaster machine. We typed the command we wanted to send to the C&C servers in a terminal in the botmaster machine. The command was processed by the Python code and sent to the C&C servers as a series of payments.

D-LNBot: Different than LNBot, we used *Core Lightning* (version 0.10.0) for the full LN nodes in D-LNBot. It is one of the implementations of LN developed by Blockstream [Blo23a]. The details of setting up a Core Lightning node that is configured to run on the C&C servers are given at our GitHub page[6]. Specifically, we explain the steps for installing *Core Lightning* and *bitcoind*, configuring *Core Lightning* and *bitcoind*, and installing the necessary *Core Lightning plugins*. Similar to the LNBot, we used Bitcoin's Testnet for our proof-of-concept development. We created 3 C&C servers at various locations in US. The channels for the C&C servers were opened manually instead of using the *autopilot* feature of LN. The number of opened channels were 3 for all C&C servers. To send the botmaster's commands, we utilized the *noise* plugin in *Core Lightning*. Basically, it enables one to attach messages to the key send payments. Since the commands are sent using a single payment, there was no need for writing a script to serialize the payments into commands like in LNBot.

In addition to setting up C&C servers on Bitcoin networks, we performed several simulations in a specially crafted time-driven simulation environment using Python.

[5] `https://lightning.engineering/api-docs/api/lnd/`

[6] `https://github.com/startimeahmet/D-LNBot`

In our setup, we simulated the D-LNBot formation phase and the command propagation in D-LNBot. The simulation environment can be found on our GitHub repository as well. Specifically, in the simulations, C&C servers joined the network at random times and attempted to discover each other based on a certain policy as explained in Section 4.2.4. For the command propagation, using the generated network, the C&C servers sent the commands to their neighbors in a P2P fashion as explained in Section 4.2.5. In these simulations, we used an up-to-date LN topology which is made publicly available [Dec22]. The dataset is a collection of LN's gossip messages that is collected by a number of LN nodes running on Bitcoin Mainnet.

4.4 Evaluation and Analysis

In this section, we present a detailed cost and delay analysis of LNBot, iLNBot, and D-LNBot. While analyzing the different implementations, it is important to note that the selection of Bitcoin Mainnet and Bitcoin Testnet dramatically affects the evaluation results. While the infrastructure they offer are exactly the same, the cost of running the botnet is different. Testnet Bitcoins do not have a monetary value and thus they can be obtained for free from the faucets. On the other hand, Mainnet Bitcoins do have a real monetary value and have to be purchased.

4.4.1 LNBot Evaluation

While it is mostly a design choice, we used Bitcoin Mainnet to evaluate the LNBot because of its bigger size. Also, as will be explained in Section 4.5, historically Mainnet had more uptime than Testnet.

Cost Analysis of LNBot Formation

We first analyze the monetary cost of forming LNBot. As noted earlier, we opened 3 channels per server. The capacity of each channel is 20,000 satoshi which is the minimum allowable channel capacity in *lnd*. Therefore, a server needs 60,000 satoshi for opening these channels. While opening the channels, there is a small fee paid to Bitcoin miners since channel creations in LN are on-chain transactions. We showed that, opening a channel in LN can cost as low as 154 satoshi on both Bitcoin Testnet[7] and the Mainnet[8].

So the total cost of opening 3 channels for a C&C server is 60,462 satoshi. While 462 satoshi is consumed as fees, the remaining 60,000 satoshi on the channels is not spent, rather it is just locked in the channels. The botmaster will get this 60,000 satoshi back after closing the channels. Therefore, funds locked in the channels are non-recurring investment cost for the formation of LNBot. Only real associated cost of forming LNBot is the channel opening fees.

Table 4.1 shows how the costs change when the number of C&C servers is increased. The increase in the cost is linear and for 100 C&C servers, the on-chain fees are only 46,200 satoshi.

Cost and Delay Analysis of Command Propagation

To assess the command propagation delay, we sent the following SYN flooding attack command to the C&C servers from the botmaster (omitting the start and end

[7]Check LNB6's channel opening transaction for instance: `https://blockstream.info/testnet/tx/fc46c99233389d24c4fd9517cd503f08265c517a6f0570d806e7cc98b7f7963b`

[8]In a similar way, check one of our mainnet nodes' channel opening transaction: `https://blockstream.info/tx/1d81b6022ff1472939c4db730ca01b82d43b616e757d799aea17ee0db6427520`

Table 4.1: Channel Opening Fees for Different Number of C&C Servers

Number of C&C Servers	**Channel Opening Fees**
10	4,620 satoshi
25	11,550 satoshi
50	23,100 satoshi
100	46,200 satoshi

characters): `sudo hping3 -i u1 -S -p 80 -c 10 192.168.1.1`

We sent this command using both of the encoding methods we proposed earlier. For Huffman coding, we compared several different base number systems. The best result was obtained by using the Quaternary numeral system, the codebook of which is shown in Table 4.2.

Table 4.2: Obtained Codebook for Huffman Coding

's'	234	'n'	233	'o'	232	'h'	231
'd'	224	'g'	223	'c'	222	'9'	221
'6'	214	'2'	213	'3'	212	'u'	211
'p'	144	'i'	143	'8'	142	'0'	141
'.'	24	'1'	12	'-'	13	'E'	4
' '	11	'S'	3				

Cost Analysis: To calculate the cost of sending a command, we need to encode each character with its corresponding satoshi value. To give an example; the command `sudo` would be encoded as (115,117,100,111) with ASCII and (2,3,4,2,1,1,2,2,4,2,3,2) with Huffman where each value represents the satoshi amount of the payment. As can be seen in Table 4.3, to send the SYN flooding attack command, the botmaster spent 2,813 satoshi using the ASCII encoding and only 215 satoshi using the Huffman coding. Table 4.3 shows how many payments were sent for each piece of the command as well as the total satoshi spent for sending the whole command for both encoding methods. While in both cases the botmaster will be reimbursed at the very end, we would like to note that the lifetime of the channels is closely

Table 4.3: Breakdown of How Many Payments are Sent for the `sudo hping3 -i u1 -S -p 80 -c 10 192.168.1.1` Command with ASCII and Huffman Encoding

Command	**Number of Payments**	
	ASCII Encoding	**Quaternary Huffman Encoding**
'sudo '	5	14
'hping3 '	7	20
'-i '	3	7
'u1 '	3	7
'-S '	3	5
'-p '	3	7
'80 '	3	8
'-c '	3	7
'10 '	3	7
'192.168.1.1'	11	26
Total Number of Payments	44	108
Total Satoshi Used	2,813	215

related with these costs. In case of the ASCII encoding, the initial funds will be spent faster and the botmaster needs to reconfigure (or rebalance) the channels for continuous operation of the botnet. In case of the Huffman coding, this is not the case as the consumption of the channel funds is much slower. So, we can see that if channel lifetime is an important factor for the botmaster, the Huffman coding could be preferred. In other words, the Huffman coding gives the botmaster the ability to perform more attacks without creating high capacity channels.

However, the situation is reverse in case of routing fees. According to our experiments in LN, the average forwarding fee for payments was 4 satoshi. Taking this into account, the total forwarding fee for sending the SYN flooding attack command for different number of C&C servers is shown in Table 4.4. The increase in the routing fees is linear for both the ASCII and Huffman coding. For 100 C&C servers, total routing fee paid is only 17,600 satoshi for ASCII while it is 43,200 satoshi for the Huffman coding. This indicates that routing fees with Huffman is 2.45 times higher

than ASCII. However, the satoshi spent for sending the actual command is 13 times less than ASCII. Thus, if the botmaster is willing to open channels with higher capacities, it is more feasible to run the botnet using the ASCII method because of the less routing fees paid.

Table 4.4: Routing Fees for Different Number of C&C Servers

Number of C&C Servers	**Routing Fees (ASCII)**	**Routing Fees (Huffman)**
10	1,760 satoshi	4,320 satoshi
25	4,400 satoshi	10,800 satoshi
50	8,800 satoshi	21,600 satoshi
100	17,600 satoshi	43,200 satoshi

Delay Analysis: The propagation time of a command is calculated by multiplying the number of payments with the average delivery time of the payments. To estimate the average delivery time, we sent 90 *key send payments* with different amounts from botmaster to our C&C servers over LN at random times and measured the time it took for payments to reach their destinations. The results are depicted in Fig. 4.5.

As shown, key send payments took 7.156 seconds on average to reach their destinations and the maximum delay was never exceeding 10 seconds. This delay varies since it depends on the path being used and the load of each intermediary node in the LN. We observed that the number of hops for the payments was 4, which helps to strengthen unlinkability of payments and endpoints in case of any payment analysis in LN.

Using an average of 7.156 seconds, the total propagation time for the ASCII encoded payments is 7.156x44=314.864 seconds while it is 7.156x108=772.848 seconds for the Huffman coding. The Huffman coding reduces the Bitcoin spent for sending the commands (not the routing fees), but increases the command sending delays.

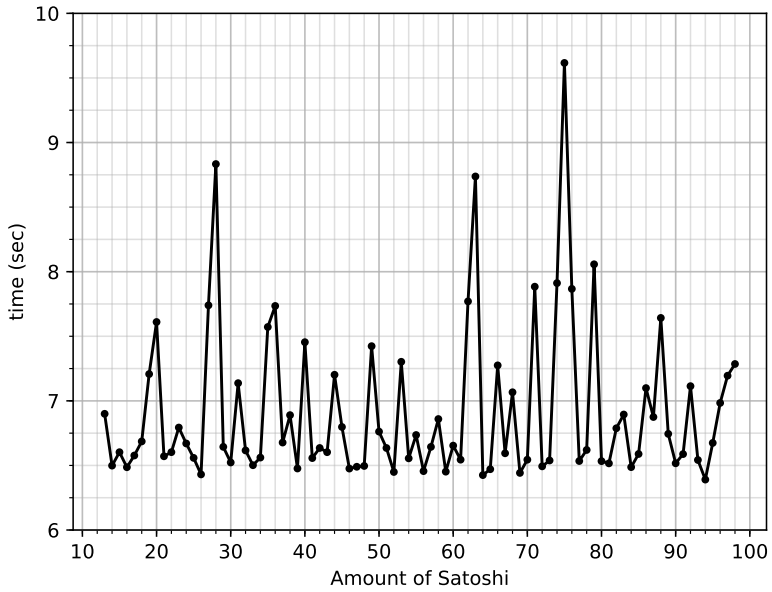

Figure 4.5: Time for *key send* payments to reach their destinations with varying satoshi.

This analysis is for sending the command to a single C&C server only. If we generalize it to n C&C servers where the command comprises of a characters; we can see that the time complexity of sending the command to all the C&C servers is in the order of $O(na)$. We will call this *total command sending delay* for later reference which basically stands for the time it takes for all the C&C servers in the botnet to receive the command. Note that, we assume that the botmaster do not *parallelize* the command sending by trying to send the command to multiple C&C servers at the same time. Instead, s/he sends the command to each C&C server sequentially.

Next, we analyze the iLNBot which uses the noise plugin for sending the commands instead of the ASCII or Huffman methods.

4.4.2 iLNBot Analysis

Using the noise plugin for command sending dramatically reduces both the delay and cost of sending the commands. For the same SYN flooding attack command, this time the botmaster just sends a single payment to each C&C server instead of sending consecutive payments for each character in the command. According to our experiments, the forwarding fee paid for a message sent using the noise plugin was 2 satoshi over an average of 30 tries. Thus, for 100 C&C servers, the total forwarding fee paid for sending the same SYN flooding attack command is only 200 satoshi. In LNBot, it was 17,600 satoshi for ASCII and 43,200 satoshi for the Huffman method. Thus, there is an improvement around 98% compared to the LNBot.

The delay of sending the command decreases in similar proportions. According to our measurements in LN, the delay of sending messages using the noise plugin was only 2.11 seconds over an average of 30 tries. Length of the messages did not affect the delay. One thing to note here is that the delay is less compared to what we measured for the LNBot (7.156 seconds) even though both utilize the key send payments. This is due to the network conditions at different times as well as the number of hops used to send the payments. If we consider the ASCII encoding case where it took 314.864 seconds to send the SYN flooding attack command, the new method is 99% faster than the ASCII method. Now, we can easily generalize the total command sending delay when there are n C&C servers in the botnet. Since the time complexity of sending the command to an individual C&C server is $O(1)$, sending it to n C&C servers is in the order of $O(n)$. This is much faster than LNBot's $O(na)$ total command sending time.

4.4.3 D-LNBot Evaluation

Evaluation of D-LNBot is slightly different than the LNBot as we propose to utilize the Bitcoin Testnet. Because of this design choice, the cost just becomes zero. Thus, we only analyze the delay of command propagation. In this direction, we first analyze the time complexity of propagating a command to all the C&C servers in D-LNBot. Then, we numerically show the total command propagation delay for varying number of C&C servers based on the simulations we performed.

The average delay of sending a message using the noise plugin was already given at the previous section which was 2.11 seconds. In D-LNBot, this corresponds to the delay of sending a command from one C&C server to another. In order to numerically calculate the total command propagation time of D-LNBot, we first need to analyze how the commands propagate among the C&C servers in a D-LNBot topology.

D-LNBot's topology forms a connected graph where each vertex is a C&C server and each edge is a logical neighbor link. To compute the time complexity of the command propagation in this graph, we first examine the two worst cases: 1) $m = 1$ and 2) $m = n$ where m is the number of active C&C servers and n is the number of C&C servers. The topologies formed for these cases are shown in Fig. 4.6 below.

In both of these cases, the time complexity of propagating a command to all the C&C servers will be $O(n)$. In $m = 1$ case, C&C servers form a straight line and the total propagation time is directly proportional to the number of C&C servers lined up as the command have to go through all the C&C servers one by one until reaching the last C&C server. Looking at the second topology where $m = n$, we see that it is the same centralized topology in LNBot and iLNBot. Thus, the time complexity of this case will also be $O(n)$ due to sequential propagation of the commands.

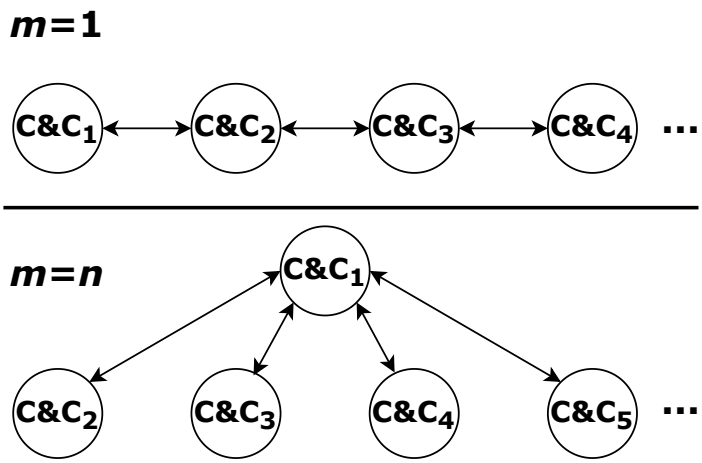

Figure 4.6: Logical topologies of the two worst cases in D-LNBot.

For $1 < m < n$ case that is of the real interest to us, we already provided two example topologies at Fig. 4.4. At first glance, the topologies hint us that the time complexity of command propagation is at least in the order of m (i.e., $\Omega(m)$). This is because we need at least m messages in any case due to minimum number of neighbors we need to reach sequentially for a node. But as seen in Fig. 4.6, the height of the tree also has a role in the number of messages since we need to use a *spanning tree* to reach every C&C server. The height of the spanning tree grows as m grows and it reduces as m decreases. This corresponds to a height of $log_m n$. Now, the first level of the tree uses m branches which already brings m messages to the total complexity. Starting from level 2 of the tree, we get a height of $log_m n$ and each level adds m more messages. This means we have $m(log_m n - 1)$ more messages added to the complexity. Thus in total, our message complexity is $m + m(log_m n - 1)$ which is in the order of $O(mlogn)$. This is faster than $O(n)$ in the average cases considering random topologies.

We can conclude that, D-LNBot has a clear advantage over LNBot on both the cost and delay of sending the commands thanks to its unique distributed design and utilization of the Bitcoin Testnet.

Simulation Results: Measuring the command sending delay between two C&C servers was straightforward however measuring the *total* command sending delay is not as straightforward since it requires a rather complicated setup to perform the experiments. Specifically, one would need to create an extensive number of machines and program each of them to properly perform the botnet formation and the command propagation operations. Additionally, since in our design, the C&C servers are created spontaneously, it is hard to realize this behavior on actual LN without actually releasing a malware in-the-wild [BAA+22, OALU22]. Instead of going this route, we performed simulations to measure the total command propagation delay in D-LNBot. Simulations also let us capture and test a wide range of cases. In this direction, we first simulated the D-LNBot formation phase in Python using the latest available LN topology at [Dec22] which consists of 13,772 nodes and 118,021 channels. In the simulation, all the required parameters such as number of active C&C servers, number of C&C servers, number of innocent nodes are set accordingly and a specific policy was set for C&Cs to discover each other. After running the simulation, we observed that all the C&C servers discovered the required number of neighboring servers and a D-LNBot was successfully formed. The resulting topologies can be reproduced by running the Python scripts in our GitHub page. As explained in Section 4.2.4, C&C servers first found the most connected nodes (i.e., innocent nodes) in the network and randomly opened a channel to one of them. Additionally, each C&C server randomly opened a channel to one of the nodes in the network to be able to route the payments even after closing its channel with the innocent node. This means that the C&C servers did not necessarily establish channels to the centralized nodes in the network which actually resulted in most of the payments being forwarded in 5 to 7 hops. And in the simulations, we assumed that each hop introduces a delay between 0.4 and 0.5 seconds which is a realistic

approximation from real LN payments we performed. To be able to simulate the multi-hop mechanism, we first calculated the shortest path between a sender and the recipient using Dijkstra's algorithm [Dij59]. Then, we created a data structure for the C&C servers to be able to carry the hop information for the payments as well as a message. In this way, C&C servers know where to forward the payment next and can deliver the message to the destination. The number of active C&C servers in the simulations were 3. Finally, to be able to compare the results with LNBot and iLNBot, we varied the number of C&C servers in the simulations.

The results of total command propagation times are shown in Table 4.5. To compare, in LNBot, sending the command to a single C&C server using the faster ASCII method took 314.864 seconds. Consequently, sending it to 10 C&C servers takes 3,148.64 seconds. This is much slower than D-LNBot's 14 seconds. In iLNBot, sending the command to each C&C server took 2.11 seconds. Thus, the total command sending delay for 10 C&C servers is 21.1 seconds which is still slower than D-LNBot as expected. The results for more C&C servers can be seen at Table 4.5. D-LNBot is faster than the other two due to its distributed topology that splits into branches. In a sense, command sending is parallelized with each branch enabling D-LNBot to achieve a faster propagation. Thus, we can conclude that D-LNBot propagates the commands to the C&C servers much faster compared to LNBot and the gap between the two opens up even more as the number of C&C servers increases.

4.4.4 Comparison with Other Similar Botnets

We also considered other existing botnets that utilize Bitcoin for their command and control. The extensive comparison of these botnet with our proposed LNBot,

Table 4.5: Total Command Propagation Time of SYN Flooding Attack Command in LNBot, iLNBot and D-LNBot for Different Number of C&C Servers

Number of C&C Servers	**D-LNBot**	**iLNBot**	**LNBot**
10	14 sec	21.1 sec	3,148.64 sec
25	36.5 sec	52.75 sec	7,871.6 sec
50	72.9 sec	105.5 sec	15,743.2 sec
100	136.5 sec	211 sec	31,486.4 sec

Table 4.6: Comparison of LNBot, iLNBot and D-LNBot with Similar Bitcoin-based Botnets

Botnet	**Network**	**Method**	**Cost**	**Delay**	**Scalability**
BOTCHAIN [PP18]	Mainnet	OP_RETURN	22,848 satoshi as of today	$\sim$ 1 hour	Low
ZombieCoin 2.0 [AMLH18]	Mainnet	OP_RETURN + subliminal channels	22,848 satoshi as of today	$\sim$ 10 seconds	Low
DUSTBot [ZZZ$^+$19]	Both Mainnet and Testnet	OP_RETURN	22,848 satoshi as of today	$\sim$ 10 seconds	Medium
ChainChannels [FAZ18]	Mainnet	ECDSA Signature	>22,848 satoshi as of today	$\sim$ 10 seconds	Medium
CoinBot [YCL$^+$20]	Either Mainnet or Testnet	OP_RETURN + website	226 satoshi	$\sim$ 6 minutes	Low
Franzoni et al. [FAD20]	Testnet	non-standard OP_RETURN	133-51,349 Testnet satoshi	$\sim$ 10 minutes	Low
LNBot	Mainnet	LN Payment	176 satoshi (per C&C)	$\sim$ 5 minutes (per C&C)	High
iLNBot	Mainnet	LN Payment	2 satoshi (per C&C)	$\sim$ 2 seconds (per C&C)	High
D-LNBot	Testnet	LN Payment	2 Testnet satoshi	$\sim$ 1.5 seconds (per C&C)	High

iLNBot and D-LNBot are presented in Table 4.6. We considered the following metrics for the comparison: 1) Which Bitcoin network is utilized by the botnet?; 2) What methods are used for command propagation?; 3) What is the cost of sending our SYN flood attack command to all the bots in the botnet?; 4) How long does it take for all the bots in the botnet to receive the SYN flood attack command from the botmaster (i.e., total command propagation time); 5) How scalable is the botnet? Low: thousands of bots, Medium: hundreds of thousands of bots, High: millions of bots.

Before interpreting the results, it is important to note that the cost of sending the commands is variable for some botnets because they utilize on-chain Bitcoin transactions whose fees change dynamically depending on the load on the Bitcoin network. At the time of writing this chapter, fee for a Bitcoin transaction with a median transaction size of 224 bytes was 22,848 satoshi[9]. For the botnets that fully run on Bitcoin's Testnet, the cost of command sending is normally zero but in order to have a quantitative comparison with the other botnets, we still mentioned the cost with *Testnet satoshi*.

As can be seen in Table 4.6, BOTCHAIN [PP18] is the worst among all with a high cost and long delay for sending the commands. ZombieCoin 2.0 [AMLH18], DUSTBot [ZZZ+19] and ChainChannels [FAZ18] are faster but still come with high costs. CoinBot [YCL+20] has much less cost due to using 1 sat/byte fee for Bitcoin transactions however, it still takes a long time to propagate the commands. Franzoni et al. [FAD20] can send the commands for free due to using Testnet however still has long delays. LNBot spends less than all previously mentioned botnets but still has long delays. iLNBot achieves much faster and cheaper command sending compared to LNBot due to the use of noise plugin. Finally, D-LNBot sends the commands even faster than iLNBot and for free ultimately being the best among all.

If we look at the scalability of these botnets, only DUSTBot [ZZZ+19] and ChainChannels [FAZ18] seem to be able to somewhat scale due to the way they propagate the commands to the bots. Others rely on blockchain confirmations of their Bitcoin transactions which greatly limits their scalability. Our proposed LNBot, iLNBot and D-LNBot on the other hand can scale freely thanks to their 2-layer architecture.

[9] `https://bitcoinfees.earn.com/`

4.5 Security & Anonymity Analysis and Countermeasures

In this section, we discuss the security properties of LNBot and D-LNBot as well as possible countermeasures to detect their activities in order to minimize their impacts.

• *Taking LN Down:* Obviously, the simplest way to eliminate LNBot and D-LNBot is to take down LN as a whole once there is any suspicion about a botnet. However, this is very unlikely due to LN being a very resilient decentralized payment channel network. In addition, today many applications are running on LN and shutting it down may cause a lot of financial loss for numerous stakeholders.

• *Resetting the Bitcoin Testnet:* Since taking LN down is not really an option, taking down the Bitcoin Network itself could be explored. Since its creation, Bitcoin Mainnet never went offline. Bitcoin Testnet on the other hand is in its third iteration, which is called Testnet3. Previous two Testnets were reset due to some technical reasons[10]. Essentially, this poses a great risk for a botnet running on the Bitcoin Testnet. When the existence of D-LNBot is discovered, it is possible that the Bitcoin developers might go the route of resetting the Testnet again. Basically, a new genesis block is generated invalidating all the previous blocks. If this happens, all LN channels created on the Bitcoin Testnet will become unusable. This action therefore will stop the D-LNBot completely.

• *Compromising and Shutting Down a C&C Server:* In our design, there are many C&C servers each of which is controlling a mini-botnet. Given the past experience with various traditional botnets, it is highly likely that these mini-botnets will be detected at some point in the future paving the way for also the detection of a C&C server. This will then result in the revelation of its location/IP address and even-

[10] `https://bitcoin.stackexchange.com/questions/36252/testnet-version-history`

tually physical seizure of the machine by law enforcement. Nevertheless, in LNBot, the seizure of a C&C server will neither reveal the identity of the botmaster nor other C&C servers since a C&C server receives the commands through onion routed payments catered with Sphinx's secure packet format, which does not reveal the original sender of the payments. Additionally, the communication between the botmaster and the C&C servers is 1-way meaning that botmaster can talk to the C&C servers, but servers cannot talk back since they do not know the LN address of the botmaster. This 1-way communication combined with the onion routed payments ensure that the identity of the botmaster will be kept secret at all times.

Note that, in LNBot, since the C&C servers hold the LN public key of the collector, it will also be revealed when a C&C server is compromised. However, since the collector's LN channels are all private, its IP address or location is not known by the C&C servers. Therefore, learning the LN public key of the collector node does not help locating it physically. However, honest LN nodes in the network can be informed of collector's LN public key and be advised to refuse opening channels to it. In this case, collector might be unable to receive funds from the C&C servers. Therefore, we offer alternatives to address this issue with the new features of LN. Best and most reliable option is to use *splicing* which enables either adding or removing funds from a channel by a single on-chain transaction without having to close the channel or create a new one. It is one of the most anticipated additions to LN because it will solve the biggest problem of LN which is the channel liquidity management. In case of C&C servers, they would *splice out* their channel funds to an on-chain output which the botmaster can spend and re-use. Currently there is an experimental implementation of splicing[11]. Thus, we rather recommend using *PeerSwap* protocol instead until splicing is fully integrated into LN. PeerSwap en-

[11] `https://github.com/ElementsProject/lightning/pull/5675`

ables swapping on-chain Bitcoins for inbound or outbound channel liquidity between a channel's peers [Blo23b]. The protocol is completely trustless, low cost and can be used today with both *Core Lightning* and *lnd*. C&C servers can *swap out* their channels funds for on-chain Bitcoins which the botmaster can spend and re-use. Eventually, we can see that taking down a single C&C server shuts down the LNBot partially resulting in less damage to victims.

In case of the D-LNBot, revealing a C&C server will also reveal its neighboring C&C servers' LN public keys and possibly IP addresses. Even though more information is being revealed, it is easier to recruit new C&C servers in D-LNBot. The reason for that is, in D-LNBot, botmaster does not create the C&C servers himself rather they are created through *malware_1* infections. To conclude, we can say that, taking down a single C&C server only reveals a few other C&C servers which is not critical considering that the number of C&C servers in D-LNBot will likely to be greater than that of LNBot. Regardless, this still takes down a portion of D-LNBot which helps reducing its impacts on the victims.

Additionally, in D-LNBot, when a C&C server is compromised, the function $f()$ and the value ξ can be revealed. In such a case, the active C&C servers' LN public keys (and possibly IP addresses) will be revealed. As explained in Section 4.2.4, there are only m active C&C servers in D-LNBot at any time, thus revelation of these hardcoded parameters will only cause revelation of m C&C servers. Older C&C servers that already closed their channels with the innocent nodes cannot be detected with this information. Thus, an observer could instead start monitoring the new channel openings in LN to identify the newly joined C&Cs. However, s/he cannot know for certain if an LN node satisfying the policy is a C&C server or a honest LN node. Because, the channel capacities satisfying the policy can also be used by honest LN nodes.

Thus, we can conclude that revelation of these parameters expose active C&C servers helping investigators take down a portion of D-LNBot.

• *Payment Flow Timing Analysis for Detecting the Botmaster:*

As explained in Section 3.3, the intermediary nodes in a payment path do not know the origin of the payment; therefore they cannot distinguish between the botmaster and a regular forwarding node on the payment path. In our tests for LNBot, we observed that our payments took 4 hops to reach the C&C servers. Therefore, payment analysis for that many hops is a challenge. However, it can help increase our chances to detect the botmaster.

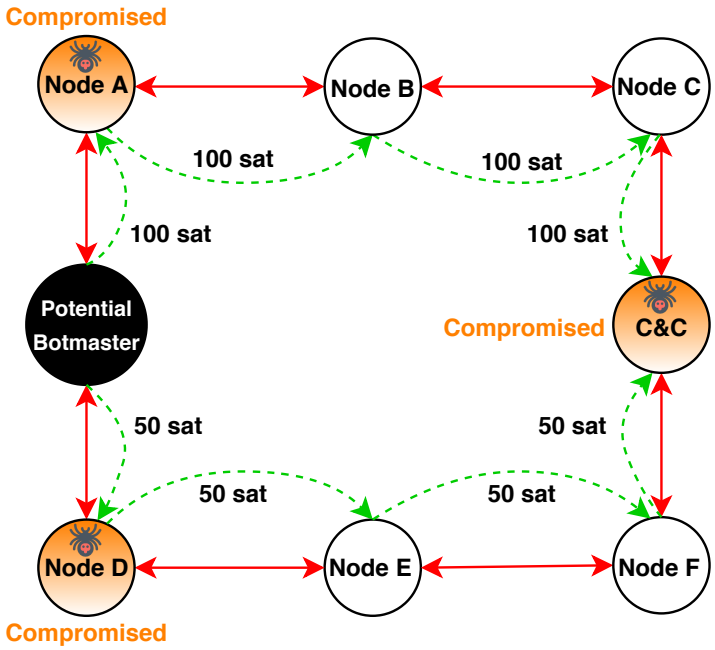

Figure 4.7: The payments that are forwarded by Node A and Node D and the payments arriving at the C&C server are monitored by an observer. Red arrows show the payment channels between the nodes and the green arrows show the flow of the payments

To further investigate this attack scenario, a topology of 8 nodes was created on Bitcoin Testnet as shown in Fig. 4.7. We assume that Node A, Node D and the C&C server are compromised and thus we monitored their payments. In this setup,

a 100 satoshi payment was sent from the botmaster to the C&C server through hops Node A, Node B, and Node C and the payment was monitored at Node A. By monitoring the node, we got the payment forwarding information shown in Fig. 4.8.


```
"forwarding_events": [
  {
    "timestamp": "1579043693",
    "chan_id_in": "1826219544504369152",
    "chan_id_out": "1826219544504434688",
    "amt_in": "101",
    "amt_out": "100",
    "fee": "1",
    "fee_msat": "1000",
    "amt_in_msat": "101000",
    "amt_out_msat": "100000"
  }]
```


Figure 4.8: The payment forwarding event captured at Node A. LN nodes keep the payment forwarding information locally and it can be accessed in JSON format with the command `lncli fwdinghistory`.

In the same way, this time a 50 satoshi payment was sent from the botmaster to the same C&C server following the hops Node D, Node E, and Node F and the payment was monitored at Node D. Similar payment forwarding information is obtained at Node D. Here, the particularly important information for us is the timestamp of the payment, and the `chan_id_in` and the `chan_id_out` arguments which represent the ID of the channels that carry the payment in and out from Node A. We can query these channel IDs to learn the public keys of the nodes at both ends of the channel by running `lncli getchaninfo chan_id`. Obtained LN public keys at Node A, in this case, belong to the potential botmaster and Node B. In the same way, LN public keys of potential botmaster and Node E is obtained at Node D. After the payment is observed at Node A, payment with the same amount was observed at the C&C server. We now correlated these two payments (i.e., timing analysis) and suspected that the sender to Node A (or D) can be a potential botmaster. Obviously, there is no guarantee for this (e.g., imagine a different topology where

the real botmaster is 2 more hops away). We need to collect more data from many compromised nodes and continue this analysis for a long time. To increase the chances, well connected LN nodes could be requested to cooperate in case of law enforcement investigation to share the timing of payments passing from them.

This analysis is applicable to D-LNBot as well. A similar topology can be created around a captured C&C server in an attempt to reveal the sender of the payments. However, the senders are already known by the C&C server; i.e., the neighboring C&C servers. So, doing this timing analysis to try to find the botmaster will be pointless as s/he is never involved in the process. Only case where this might be useful is applying the analysis to the C&C server that initially receives the command from the botmaster. However, even in that case, C&C server will only receive a single payment for each command which will make the timing analysis impracticable as the analysis requires the C&C server to receive a number of payments to possibly provide useful information.

• *Poisoning Attack:* In LNBot, another effective way to attack the botnet is through message poisoning. Basically, once a C&C server is compromised, its LN public key will be known. Using the public key, one can send payments to the C&C server at the right time to corrupt the messages sent by the botmaster. There is currently no authentication mechanism that can be used by the botmaster to prevent this issue without being exposed. Recall that the commands are encoded into a series of payments and when an attacker sends a different character during a command transmission, it will corrupt the syntax and thus eventually the command will not have any effect on the corresponding mini-botnet. The right time will be decided by watching the payments and packets arriving at the C&C server. The disadvantage of this, however, is that one needs to pay for those payments. Nonetheless, this can be an effective way to continue engaging with the botmaster for

detection purposes rather than just shutting down the C&C server while rendering any attack impossible.

iLNBot and D-LNBot are not vulnerable to this attack because of the authentication mechanism that exists in the noise messages as explained in Section 4.2.5. Even if somehow an attacker gets the LN public keys of the C&C servers and initiates commands to them using the noise plugin, the C&C servers will realize that the commands are coming from an unknown node so, they can just ignore them. Additionally, an attacker cannot poison the commands as the commands are sent with a single payment. Rather, the attacker can only try to execute his/her own commands on the C&C servers which will not be possible due to the authentication mechanism.

• *Analysis of On-chain Transactions:* Another possible attack vector to both LNBot and D-LNBot is to analyze the on-chain Bitcoin transactions of the C&C servers (i.e., channel opening and closing transactions). For such forensic analysis, the Bitcoin addresses of the C&C servers have to be known. As with many other real-life botnets, botmasters generally use Bitcoin mixers to hide the source of the Bitcoins. Usage of such mixers makes it very challenging to follow the real source of the Bitcoins since the transactions are mixed between the users using the mixer service [vWOvD18]. Even though, the chances of finding the identity of the botmaster through this analysis is low, it can provide some useful information to law enforcement.

• *Packet Inspection:* Finally, LN payments could also be analyzed with packet inspection tools such as Wireshark. However, this will prove to be not very useful because all LN payments are encapsulated into onion packets as explained in Section 3.3. Any information that might be included in the payments such as messages, commands or instructions cannot be decrypted without having access to the associated private keys.

CHAPTER 5

LNGATE^2: SECURE BIDIRECTIONAL IOT MICRO-PAYMENTS USING BITCOIN'S LIGHTNING NETWORK AND THRESHOLD CRYPTOGRAPHY

Bitcoin has emerged as a revolutionary payment system with its decentralized ledger concept; however it has significant problems such as high transaction fees and low throughput. Lightning Network (LN), which was introduced much later, solves most of these problems with an innovative concept called off-chain payments. With this advancement, Bitcoin has become an attractive venue to perform micro-payments which can also be adopted in many IoT applications (e.g., toll payments). Nevertheless, it is not feasible to host LN and Bitcoin on IoT devices due to the storage, memory, and processing restrictions. Therefore, in this chapter, we propose a secure and efficient protocol that enables an IoT device to use LN's functions through an untrusted gateway node. Through this gateway which hosts the LN and Bitcoin nodes, the IoT device can open & close LN channels and send & receive LN payments. This delegation approach is powered by a threshold cryptography based scheme that requires the IoT device and the LN gateway to jointly perform all LN operations. Specifically, we propose thresholdizing LN's Bitcoin public and private keys as well as its public and private keys for the new channel states (i.e., commitment points). We prove with a game theoretical security analysis that the IoT device is secure against collusion attacks. We implemented the proposed protocol by changing LN's source code and thoroughly evaluated its performance using several Raspberry Pis. Our evaluation results show that the protocol; is fast, does not bring extra cost overhead, can be run on low data rate wireless networks, is scalable and has negligible energy consumption overhead. To the best of our knowledge, this is the first work that implemented threshold cryptography in LN.

5.1 System & Threat Model

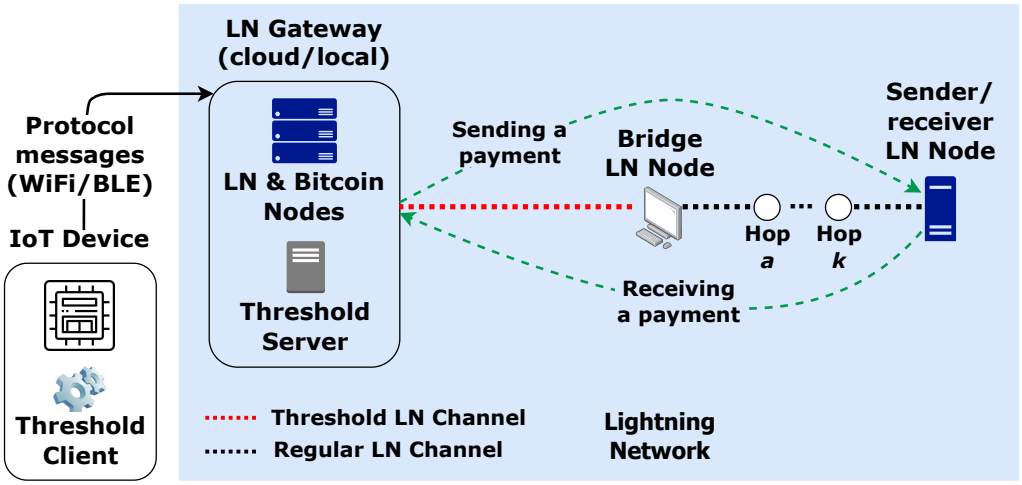

Figure 5.1: Illustration of the system model.

5.1.1 System Model

There are four main entities in our system which are **IoT device**, **LN gateway**, **Bridge LN node**, and **Sender/receiver LN node** as shown in Fig. 5.1. We also show other tools and intermediary devices; **Threshold client**, **LN gateway's LN and Bitcoin nodes**, **Threshold server**. IoT device wants to pay the receiver LN node for the goods/services. IoT device can also receive payments if needed such as a refund or a payment from another LN node (sender LN node). In other words, payments are bidirectional meaning that the IoT device can both send and receive payments. We assume that the owner of the IoT device also operates the device for these transactions. Any third party operation of the IoT devices is also possible but this may raise business privacy issues among the owner and the operator which is beyond the scope of this work. The LN gateway can be hosted on the cloud or locally depending on the use case scenario. It provides services to the IoT device by running the required full Bitcoin and LN nodes and is *incentivized by the fees the IoT device pays in return*. The LN gateway also runs a threshold server that

communicates with the threshold client installed at the IoT device when required. This client/server setup enables the 2-party threshold ECDSA operations. Bridge LN node is the node to which the LN gateway opens a channel when requested by the IoT device. Through the bridge LN node, IoT device's payments are either routed to a receiver LN node specified by the IoT device or delivered to the IoT device from a sender LN node on the Internet.

We assume that the IoT device and the LN gateway do not go offline in the middle of a process such as sending or receiving a payment. IoT device can be offline for the rest of the time.

5.1.2 Threat Model

We assume that the communication between the IoT device and the LN gateway is secured using TLS-like mechanisms. Based on our system model and application, possible adversaries to the system are the LN gateway and the bridge LN node which are assumed to be malicious. Therefore, we consider the attacks that would potentially be performed by these actors as well as the ones related to the specific processes of our payment application for the IoT devices. For instance, we assume that, both of these actors can be selfish in the sense that they can send old channel states to the Bitcoin blockchain in an attempt to cheat. We also assume that the LN gateway and the bridge LN node can collude with each other for deceiving the IoT devices. Furthermore, these nodes can also deviate from the proposed protocol descriptions to make monetary benefits.

Note that there may be also external attacks to Bitcoin's consensus mechanism and transactions independent from our approach i.e., not specific to our application. For instance, 51% attack enables external adversaries to gain control of the

blockchain [SSN+19]. Similarly, double spending attack tries to enable spending of the same currency at least two times [KAC12]. There are also attacks to LN by congesting the channels or the layer-1 [MZ21, SS23]. Since mitigations to these known attacks are already analyzed in previous studies [SMG19, NWG+20] or in the same papers where the attacks are proposed (e.g., for LN), we assume that our protocol will not be impacted from these attacks. Based on these assumptions, we consider the following attacks to our system:

- **Threat 1: Collusion Attacks:** The LN gateway and the bridge LN node can collude with each other to steal money from the IoT device.
- **Threat 2: Stealing IoT Device's Funds:** The LN gateway can steal IoT device's funds that are committed to the channel by 1) sending them to other LN nodes; 2) broadcasting revoked states and; 3) colluding with the bridge LN node.
- **Threat 3: Ransom Attacks:** The LN gateway can deviate from the protocol after opening a channel for the IoT device and not execute IoT device's requests (i.e., uncooperative LN gateway). Then, it can ask the IoT device to pay a ransom before executing the payment sending/receiving or channel closing operations.

5.2 Proposed Protocol Details

This section explains the details of the protocol that includes the channel opening, sending a payment, receiving a payment and channel closing. As mentioned in Section 3.1, we propose modifications to LN's BOLT #2 which are shown throughout the protocol descriptions. More importantly, LN's signing mechanism is modified and replaced with a (2,2)-threshold scheme that is utilized by the IoT device and the LN gateway. In addition to that, we propose changes to LN's commitment

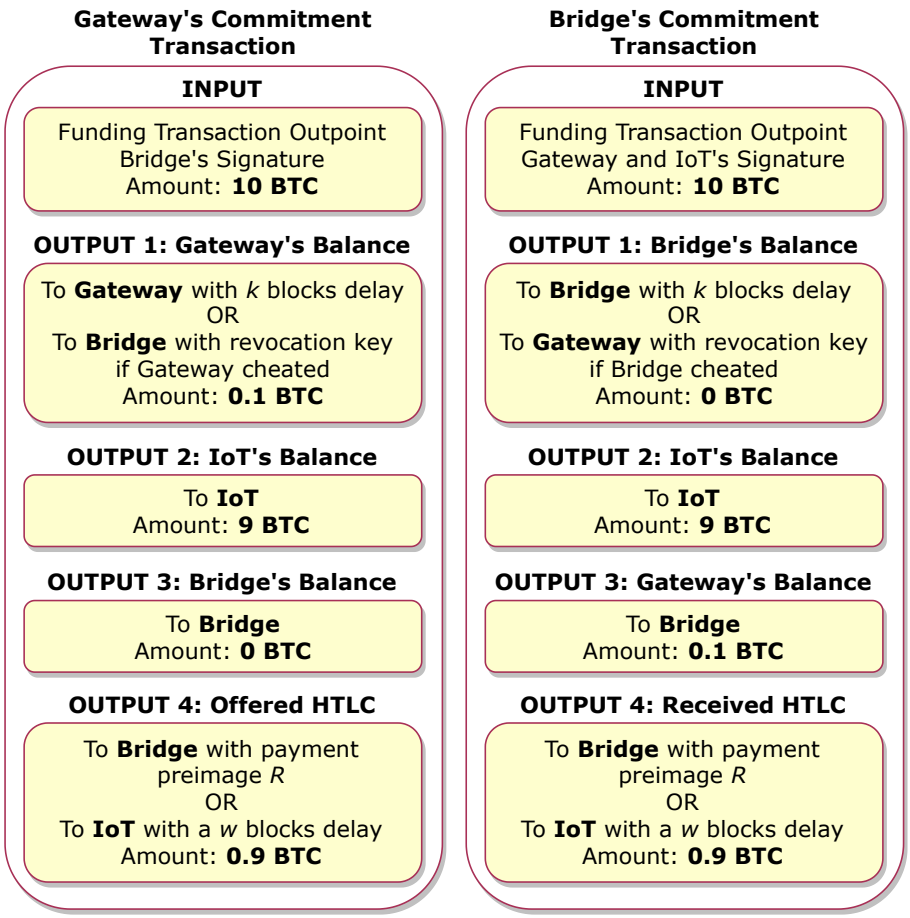

Figure 5.2: Depiction of the proposed commitment transactions for the LN gateway and the bridge LN node. These commitment transactions are generated after the following operations: 1) A channel with 10 BTC capacity was opened, 2) IoT requested sending 1 BTC to a destination, 3) Gateway charged the IoT a service fee of 0.1 BTC for this payment (the fee in real life would be much less).

transactions to accompany IoT device's funds in the channel which is explained next.

5.2.1 Modifications to LN's Commitment Transactions

Introduction of the IoT device requires some modifications to LN's commitment transactions as there are now 3 channel parties instead of 2. Since each channel party has a separate balance in the channel, they have to have an output in the

commitment transactions reflecting their balance. An illustration of LN's original commitment transaction was given in Fig. 3.4.

Our proposed modified version of LN's commitment transactions are shown in Fig. 5.2. As can be seen, there is an extra output for the IoT device in both versions of the commitment transaction. This output is not time-locked nor conditional unlike other outputs as the IoT device cannot be punished because of cheating attempts from other channel parties. This essentially protects IoT device's funds in the channel. Apart from that, the LN gateway's and the bridge LN node's outputs are regular time-locked outputs and are spendable by the counterparty in case of a cheating attempt.

5.2.2 Channel Opening Process

The IoT device is not able to open a channel by itself as it does not have access to LN nor Bitcoin network. Therefore, we enable the IoT device to securely initiate the channel opening process through the LN gateway and jointly generate signatures with it using the (2,2)-threshold scheme. This means that the LN's current channel opening protocol needs to be modified according to our needs.

All the steps for the channel opening protocol which includes the default LN messages and our additions are depicted in Fig. 5.3. We explain the protocol step by step below:

- **IoT Channel Opening Request**: IoT sends an *OpenChannelRequest* message (Message #1 in Fig. 5.3) to the LN gateway to request a payment channel to be opened to a bridge LN node. This message has the following fields: *Type: OpenChannelRequest, Channel Capacity, Bridge LN Node. Channel Capacity* is specified by the IoT device and this amount of Bitcoin is taken from IoT device's

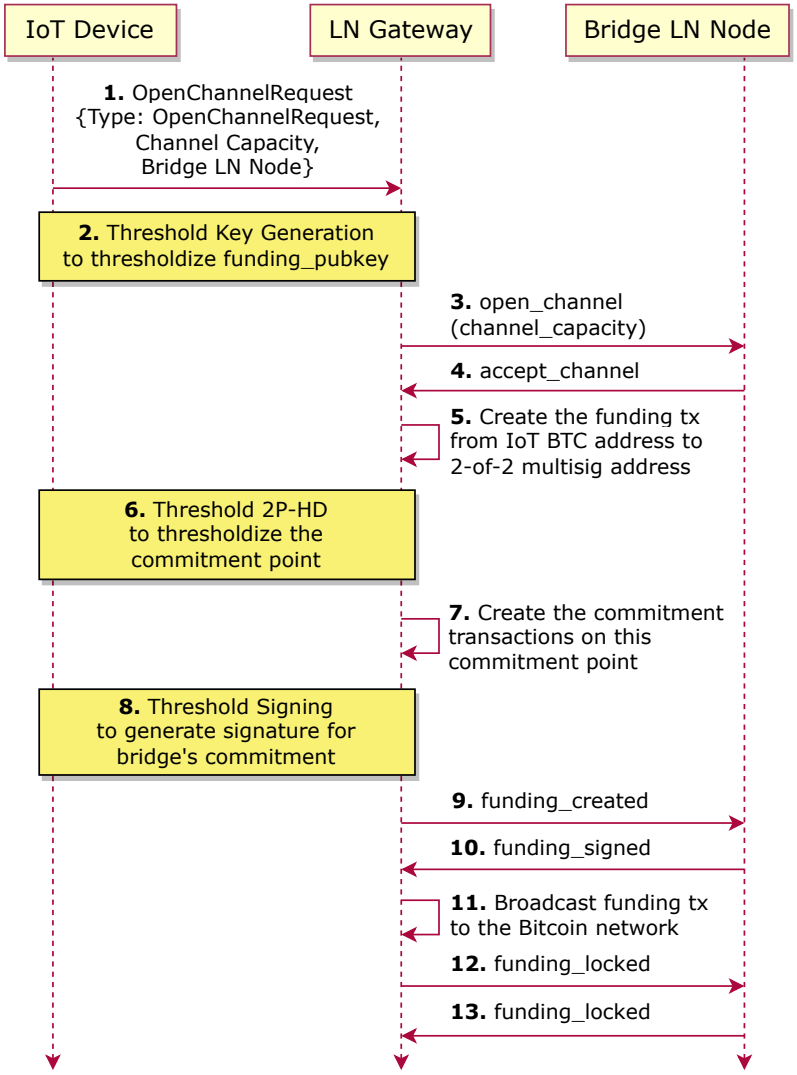

Figure 5.3: Protocol steps for opening a channel.

Bitcoin address as will be explained in the next steps. Here, we opt to let the IoT device choose the *bridge LN node* since letting the LN gateway choose the bridge LN node might not be secure as will be shown in Section 5.3.1. IoT device can make this choice by accessing API services that provide public LN information[1].

- **Channel Opening Initiation**: Upon receiving the request from the IoT device, the LN gateway initiates the channel opening process by connecting to the

[1] e.g., `https://amboss.space/`

bridge LN node specified by the IoT device. Before initiating the channel opening process by sending an *open_channel* message, we propose the LN gateway to perform a (2,2)-threshold key generation with the IoT device to **thresholdize** its *funding_pubkey* (#2 in Fig. 5.3). The *funding_pubkey* is a Bitcoin public key and both channel parties have their own. This process replaces the LN gateway's Bitcoin public and private keys with a threshold public/private key pair that is jointly computed between the IoT device and the LN gateway. In this way, the LN gateway cannot spend the funds the IoT device is committing to the channel without the IoT device's authorization. After this step, the LN gateway sends the *open_channel* message (#3 in Fig. 5.3) to the bridge LN node which includes the thresholdized *funding_pubkey*. The *channel capacity* specified by the IoT device is also sent with this message. After this step, the bridge LN node responds with an *accept_channel* message (#4 in Fig. 5.3) to acknowledge the channel opening request of the LN gateway. Note that, *open_channel* and *accept_channel* are default BOLT #2 messages.

- **Creating the Transactions**: Now that the channel parameters are agreed on, the LN gateway can create a funding transaction from IoT device's Bitcoin address to the 2-of-2 multisignature address of the channel. Since the input to the funding transaction is from the IoT device's BTC address, IoT device can also pay the on-chain fee for this transaction[2]. At this step, the LN gateway also creates the commitment transactions for itself and the bridge LN node (#7 in Fig. 5.3). Here, we propose to compute the first *commitment point* jointly between the IoT device and the LN gateway. Commitment points are used to derive revocation keys

[2]Custom funding transactions like this can be constructed by first creating a partially signed Bitcoin transaction (PSBT), then adding the inputs externally and finalizing it in LN. See: `https://docs.corelightning.org/reference/lightning-openchannel_init`

and they are unique for each channel state. Thus, thresholdizing the commitment points prevents the IoT device and the LN gateway from single-handedly revealing the revocation key before the channel state is updated. For this, we propose using a (2,2)-threshold child key derivation process (#6 in Fig. 5.3) also known as, 2P-HD [Tea18]. 2P-HD allows the derivation of child keys from the master key that was already generated with (2,2)-threshold key generation earlier (#2 in Fig. 5.3). We propose using 2P-HD over (2,2)-threshold key generation because of its efficiency.

- **Exchanging Signatures**: Now, the LN gateway needs to send the signature for bridge LN node's version of the commitment transaction to the bridge LN node. For this, the LN gateway and the IoT device jointly generate the signature in a (2,2)-threshold signing (#8 in Fig. 5.3). After signing is done, the LN gateway sends the signature to the bridge LN node along with the outpoint of the funding transaction in a *funding_created* message (#9 in Fig. 5.3). Learning the funding outpoint, the bridge LN node is now able to generate the signature for the LN gateway's version of the commitment transaction and sends it over to the LN gateway in *funding_signed* message (#10 in Fig. 5.3).
- **Broadcasting the Transaction to the Bitcoin Network**: After the LN gateway receives the *funding_signed* message from the bridge LN node, it must broadcast the funding transaction to the Bitcoin network (#11 in Fig. 5.3). Then, the LN gateway and bridge LN node should wait for the funding transaction to reach a specified number of confirmations on the blockchain (generally 3 confirmations). After reaching the specified depth, the LN gateway and the bridge LN node exchange *funding_locked* messages which finalizes the channel opening (#12-13 in Fig. 5.3).

5.2.3 Sending a Payment

Similar to the channel opening, we incorporate the IoT device in threshold operations to authorize a payment sending. The same threshold schemes are utilized and LN's current payment sending protocol is modified. The steps of the proposed payment sending protocol is depicted in Fig. 5.4 and elaborated below:

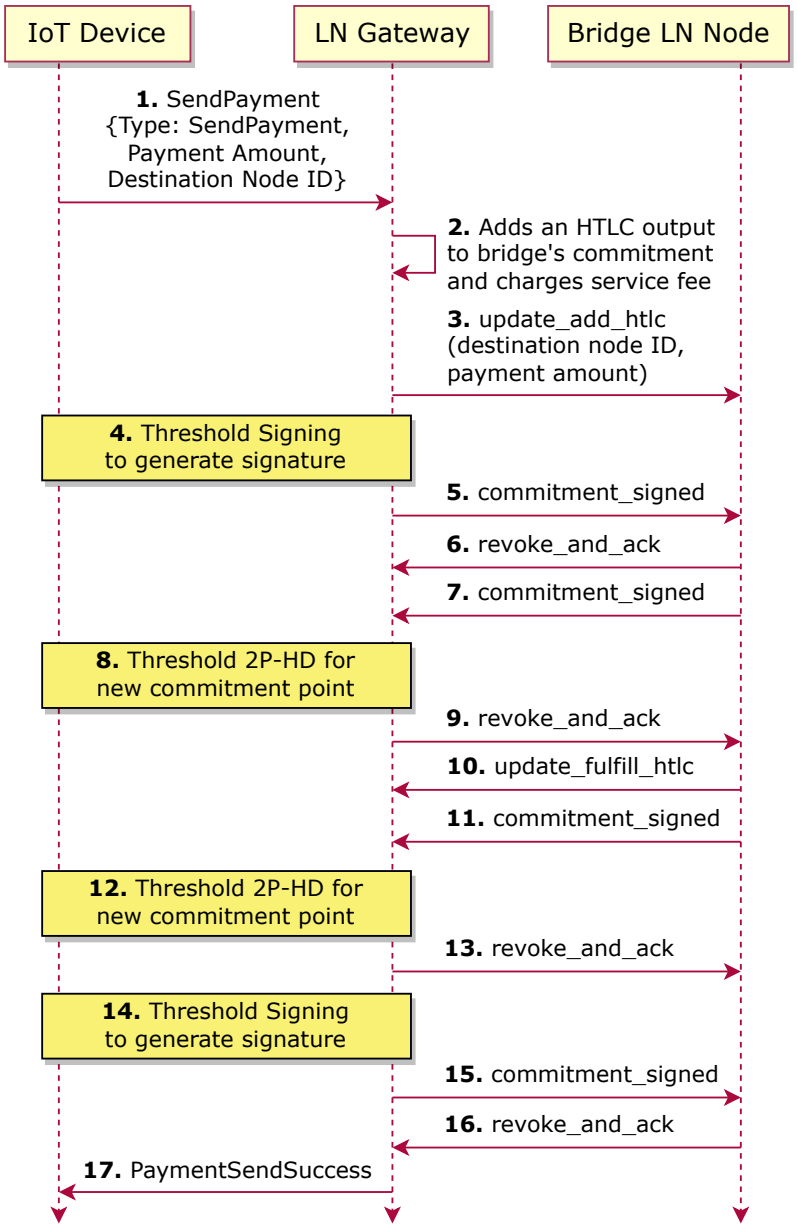

Figure 5.4: Protocol steps for sending a payment.

- **Payment Sending Initiation**: To request a payment sending, IoT device sends a *SendPayment* message (#1 in Fig. 5.4) to the LN gateway. This message has the following fields: *Type: SendPayment, Payment Amount, Destination Node ID*. Here, we assume that the *Destination Node ID* is either interactively provided to the IoT device in some form (i.e., QR code) by the vendor (i.e., toll gate) just before the payment or it is known by the IoT device in advance.

- **Payment Processing at the LN Gateway**: Upon receiving the request, the LN gateway adds an HTLC output to bridge LN node's version of the commitment transaction (#2 in Fig. 5.4). When preparing the HTLC, the LN gateway *deducts a certain amount of fee* from the real payment amount the IoT device wants to send to the destination. Therefore, the remaining Bitcoin is sent with the HTLC. This fee is taken to incentivize the LN gateway to continue serving the IoT devices. The LN gateway then sends an *update_add_htlc* message (#3 in Fig. 5.4) to actually offer to the HTLC to the destination LN node which is first received by the bridge LN node and then other nodes on the payment path (destination LN node is not shown in Fig. 5.4 for simplicity). Here, the *destination node ID*, specified by the IoT device, is embedded into the *onion routing packet* which is sent with the *update_add_htlc* message.

- **1st Commitment Round**: In LN, HTLCs always require two rounds of *commitment_signed* and *revoke_and_ack*. First round is for invalidating the old channel state right before the HTLC is attached to the channel. The second one is to fulfill the HTLC to remove it from the channel. Thus, at this point, the LN gateway will initiate the first round by sending a *commitment_signed* message to the bridge LN node. For that, it jointly generates a signature with the IoT device in a (2,2)-threshold signing (#4 in Fig. 5.4). The HTLC signature is also jointly generated at this step. Once the commitment and HTLC signatures are generated, they are

sent to the bridge LN node in the *commitment_signed* message (#5 in Fig. 5.4). These signatures will enable bridge LN node to spend the new commitment transaction and the HTLC output. The bridge LN node responds to this message by first sending a *revoke_and_ack* then a *commitment_signed* message (#6-7 in Fig. 5.4). Symmetrically, now the LN gateway will send a *revoke_and_ack* message but before that we propose the LN gateway and the IoT device to thresholdize the new commitment point in a 2P-HD (#8 in Fig. 5.4). Once this is done, the LN gateway sends the *revoke_and_ack* message (#9 in Fig. 5.4).

- **Fulfilling the HTLC**: Now, the next step is for bridge LN node to fulfill the HTLC with a similar symmetric *commitment_signed* & *revoke_and_ack* round. Thus, it sends an *update_fulfill_htlc* message to the LN gateway (#10 in Fig. 5.4) then a *commitment_signed* message (#11 in Fig. 5.4) to initiate the second round of commitments. This will be followed by the LN gateway sending a *revoke_and_ack* message. However, the LN gateway first performs a 2P-HD with the IoT device to thresholdize the new commitment point (#12 in Fig. 5.4). After sending the *revoke_and_ack* message (#13 in Fig. 5.4), the next step is to send the *commitment_signed* message. To generate the signature, the LN gateway performs a (2,2)-threshold signing (#14 in Fig. 5.4) with the IoT device then sends the *commitment_signed* message (#15 in Fig. 5.4). Finally, the bridge LN node replies with a *revoke_and_ack* message to irrevocably fulfill the HTLC (#16 in Fig. 5.4).

- **Notifying IoT**: Now that the payment is successfully sent, the LN gateway can notify the IoT device of the successful payment by sending a *PaymentSendSuccess* message (#17 in Fig. 5.4).

5.2.4 Receiving a Payment

Receiving payments on the channel is a bit different than other channel operations as it does not require the IoT device to send a request to the LN gateway. Rather, an LN node on the Internet initiates a payment to the IoT device which needs to be received on the channel that the LN gateway opened for it. Normally, when sending a payment on LN, it is enough to only specify the recipient's LN public key. If the recipient has multiple channels that can receive the payment, then the payment might end up at any of them. The exact channel that will receive the payment is internally decided by LN's routing algorithm. For our context, since the LN gateway has multiple channels each of which might be serving different IoT devices, receiving a payment on a specific channel becomes an important problem to tackle. In this direction, a sender LN node can force its payment to take a predetermined path which is also known as the *source routing*. By querying the possible routes to the recipient node, a sender LN node can decide all the channels to use before sending the payment. We propose that the LN gateway *does not charge a service fee* for receiving payments. The steps of the proposed protocol is shown in Fig. 5.5 and explained in detail below:

- **Calculating a Route:** We assume that the sender LN node already knows to which IoT device to initiate the payment and the channel ID belonging to that IoT device (i.e., the channel the LN gateway opened to the bridge LN node for this specific IoT device). Knowing the channel ID, the sender LN node can calculate a route from its LN node to the LN gateway's LN node where the channel belonging to the IoT device is the last channel on the route. After calculating the route, the sender LN node can prepare a *key send* payment that will use this specific route.

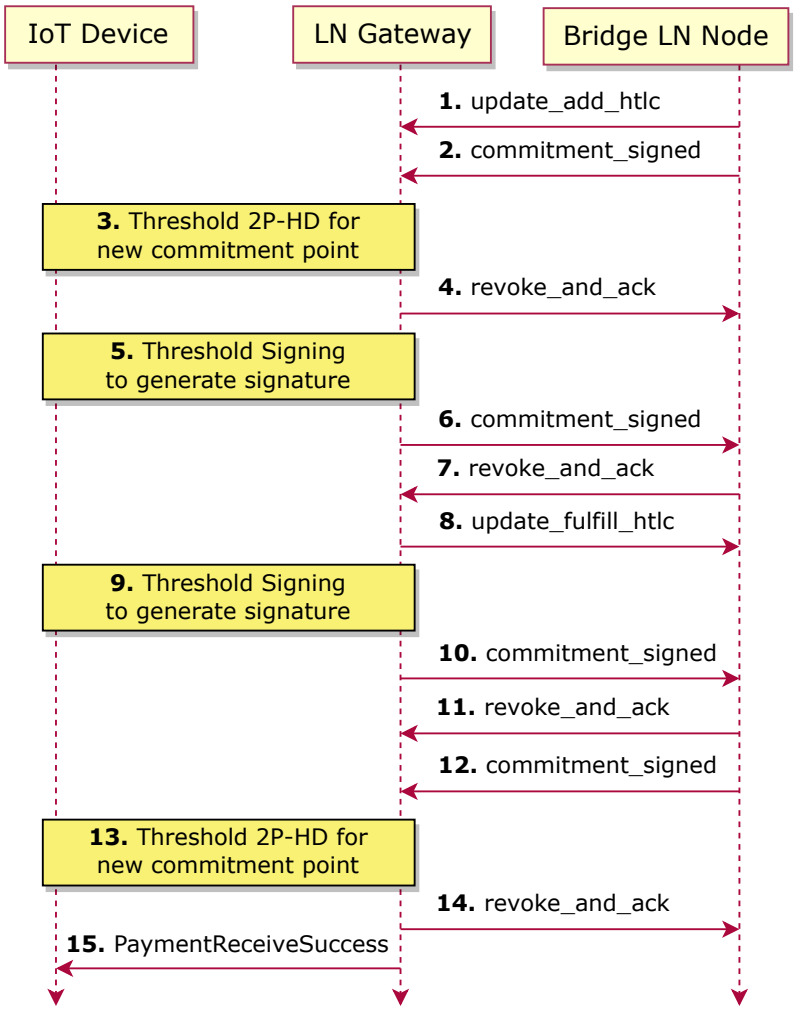

Figure 5.5: Protocol steps for receiving a payment.

- **1st Commitment Round:** The sender LN node can initiate the payment by sending an *update_add_htlc* message (#1 in Fig. 5.5) to the LN gateway which will be relayed to all the nodes on the payment path including the bridge LN node (sender LN node is not shown in Fig. 5.5 for simplicity). At this stage, similar to the payment sending case, there are going to be two rounds of *commitment_signed* and *revoke_and_ack*. The bridge LN node initiates the first round by sending a *commitment_signed* message to the LN gateway to commit the initial changes on the channel (#2 in Fig. 5.5). Here, before sending a *revoke_and_ack* message, the LN gateway performs a threshold 2P-HD (#3 in Fig. 5.5) with the IoT device to

thresholdize the next commitment point. Then, it sends the *revoke_and_ack* message (#4 in Fig. 5.5). Now, the LN gateway needs to send a *commitment_signed* message to the bridge LN node. Thus, to generate the signature, it performs a (2,2)-threshold signing with the IoT device (#5 in Fig. 5.5). Then, the *commitment_signed* message is sent (#6 in Fig. 5.5) and a *revoke_and_ack* message is received (#7 in Fig. 5.5) from the bridge LN node.

- **Fulfilling the HTLC:** Messages #2-7 made sure that the old channel state is invalidated so the bridge LN node cannot pretend the HTLC never existed. Now, the next step is to fulfill the HTLC with a similar symmetric *commitment_signed* & *revoke_and_ack* round. Thus, the LN gateway first sends an *update_fulfill_htlc* message (#8 in Fig. 5.5). Then, the LN gateway and the IoT device generate a signature in a (2,2)-threshold signing (#9 in Fig. 5.5) which is sent to the bridge LN node in a *commitment_signed* message (#10 in Fig. 5.5). Bridge LN node responds by first sending a *revoke_and_ack* message (#11 in Fig. 5.5) then a *commitment_signed* message (#12 in Fig. 5.5). Finally, before sending the *revoke_and_ack* message, the LN gateway again performs a threshold 2P-HD with the IoT device to thresholdize the next commitment point (#13 in Fig. 5.5). It then sends the *revoke_and_ack* message (#14 in Fig. 5.5) which fulfills the HTLC irrevocably.

- **Notifying IoT:** Now, it is a good time for the LN gateway to let the IoT device know that the payment is successfully received. Thus, it sends a *PaymentReceiveSuccess* message to the IoT device.

5.2.5 Channel Closing Process

A channel in LN is closed either unilaterally by one of the parties broadcasting its most recent commitment transaction to the blockchain or closed mutually by both parties agreeing on the closing fee. In our case, all 3 parties of the channel namely; the IoT device, the LN gateway, and the bridge LN node can close the channel. We explain all three cases separately below:

IoT Device Channel Closure

When the IoT device would like to close the channel, it follows the proposed protocol below.

- **IoT Device Channel Closing Request:** The IoT device sends a *ChannelClosingRequest* message to the LN gateway.
- **Mutual Close:** The LN gateway has two options to close the channel which are *unilateral* or *mutual* close. In mutual close, the LN gateway and the bridge LN node first exchange *shutdown* messages and then start negotiating on the channel closing fee. For this, they start exchanging *closing_signed* messages. This message includes the offered fee and offering party's signature. Thus, each time the LN gateway sends a *closing_signed* message, it has to perform a (2,2)-threshold signing with the IoT device to generate the signature. Once the closing fee is agreed upon, the closing transaction is broadcast to the blockchain by the LN gateway.
- **Unilateral Close:** In the unilateral close case, the LN gateway just broadcasts its most recent commitment transaction to the Bitcoin network after getting it (2,2)-threshold signed with the IoT device. Once the broadcast transaction is mined, the channel is closed and everyone's funds in the channel settle in their respective Bitcoin addresses.

We propose the on-chain fee for both cases to be paid by the IoT device since the channel closing was requested by the IoT device. The fee is deducted from the IoT device's Bitcoin address.

LN Gateway Channel Closure

The LN gateway can also initiate the channel closing if it wants to close IoT device's channel for any reason. The steps are very similar to IoT device channel closure case and explained below:

- **LN Gateway Channel Closing Request:** The LN gateway sends a *ChannelClosingRequest* message to the IoT device to show its intention to close the channel.
- **Closing the Channel:** The LN gateway can close the channel unilaterally or mutually. For either case, it needs the IoT device to participate in a (2,2)-threshold signing. The steps for closing the channel are exactly the same of the IoT device channel closure case explained above. The only difference is that, since now the channel closing is requested by the LN gateway, the on-chain fee is paid by the LN gateway by deducting the fee from its Bitcoin address.

Bridge LN Node Channel Closure

The bridge LN node can close the channel unilaterally or mutually. A mutual close from the bridge LN node will trigger a fee negotiation phase with the LN gateway which involves exchanging *closing_signed* messages. Since this requires the IoT device to be online and participate in (2,2)-threshold signing with the LN gateway, the bridge LN node might have to close the channel unilaterally if the IoT device is not online at the time of the mutual close attempt. For the unilateral close, the

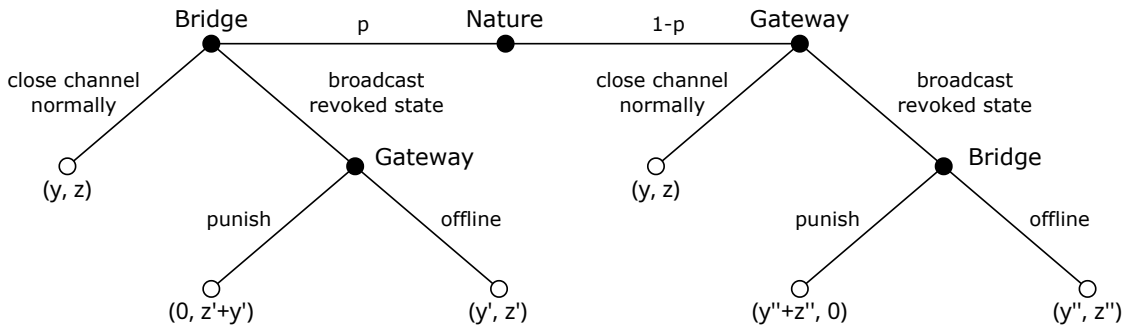

Figure 5.6: Extensive form of the behavioral game for channel closures between the LN gateway and the bridge LN node.

bridge LN node can just broadcast its most recent commitment transaction to the blockchain.

Behavioral Analysis of Channel Closures

In our preliminary version of this work [KMS+21], the protocol was only capable of unidirectional payments. With unidirectional payments, the LN gateway and the bridge LN node cannot make profit by broadcasting revoked states or colluding with each other. However, enabling bidirectional payments where the IoT device can also receive payments from other nodes in LN requires revisiting the previous analysis. The problem is, if the IoT device receives a payment, it will have some old states in which it has less money. Thus, the LN gateway or the bridge LN node can publish these old channel states to the blockchain to cheat and profit. To better illustrate these cases and have a formal analysis, we will use a *sequential game* approach using *game theory* and present the actions of each party that leads to various outcomes. Then, we will present the *Nash equilibrium* for each case using *backward induction*. We assume that all players act rationally to maximize their profits.

Game Model: The game theoretical model in an extensive form, i.e., tree-form, is shown in Fig. 5.6 and set up as follows: There are two players in this game: The

LN gateway (Gateway) and the bridge LN node (Bridge). Both the Gateway and Bridge can start the game.

With probability p, Bridge starts the game. So, at its first decision node, Bridge has to make a decision to close the channel normally or broadcast a revoked state to the blockchain. If Bridge plays to close the channel normally, the game ends at this terminal node with the payoffs of y BTC and z BTC for Bridge and Gateway, respectively (i.e., (y, z) BTC). Furthermore, we assume that the IoT device (IoT) will have x BTC for this case. Here, we can write the following equation:

$$x + y + z = C \tag{5.1}$$

where C is the channel balance.

If Bridge chooses to broadcast a revoked state, the Gateway gets to play at its first decision node in this game. At this node, normally Gateway will punish the Bridge as that is the way the LN protocol works. As explained before, if an LN node broadcasts an old state, its funds will be automatically swept by the counterparty. However, since we are examining the behavior where Gateway and Bridge are trying to steal funds from the IoT, there is a chance that Gateway will go offline before Bridge broadcasts the old state. Or, it might also happen that Gateway goes offline due to a power/Internet outage which Bridge can see and try to exploit. Thus, at this node, Gateway can either punish the Bridge or be offline.

If Gateway plays to punish the Bridge, Bridge's funds in this old state will be taken by the Gateway as proposed in our protocol description in Section 5.2.1. If the channel balances of IoT, Bridge and Gateway are (x', y', z') BTC respectively in this old state, Gateway will have $z' + y'$ BTC after punishing the Bridge. Here, we can write the following equations:

$$
\begin{aligned}
x' + y' + z' &= C \\
y' &> y \\
z' &\leq z \\
x' &< x
\end{aligned}
\tag{5.2}
$$

because we know that Bridge has more funds in the broadcast revoked state and Gateway either has the same amount of funds or less. Gateway cannot have more funds in an old state because its balance only increases when IoT sends payments. Gateway does not charge fees when IoT receives payments on the channel. Hence, the game ends at this terminal node with the payoffs (0, $z' + y'$) BTC.

If Gateway plays to go offline instead, Bridge will end up with some extra funds. Thus, the end game payoffs will be (y', z') BTC.

Game can also be started by the Gateway with probability $1 - p$. Normally, Gateway does not have an incentive to broadcast a revoked state because it does not have any old states in which it has more funds. But, we still analyze it for the sake of completeness. In its decision node, Gateway decides to whether close the channel normally or broadcast an old state. If Gateway chooses to close the channel normally, the game ends at this terminal node with the same payoffs as before; (y, z) BTC. If Gateway chooses to broadcast an old state where the channel balances are (x'', y'', z'') BTC for IoT, Bridge and Gateway respectively, Bridge will have the options of whether to punish the Gateway or be offline. Here, similar as before, we can write the following equation:

$$
x'' + y'' + z'' = C \tag{5.3}
$$

If Bridge plays to punish the Gateway, Gateway's funds in this old state will be taken by Bridge as proposed in our protocol description in Section 5.2.1. Thus,

Bridge will have $y'' + z''$ BTC after punishing the Gateway. Here, we can write the following equations:

$$\begin{aligned} y'' &> y \\ z'' &\leq z \\ x'' &< x \end{aligned} \tag{5.4}$$

One should note that, in this node, Gateway will get zero payoff. Thus, the game ends at this terminal node with the payoffs $(y'' + z'', 0)$ BTC.

If Bridge plays to be offline instead; unlike the previous scenario, Gateway will not end up with some extra funds. Instead, Bridge ends up with some extra funds. In that case, the end game payoffs will be (y'', z'') BTC.

Equilibrium Analysis:

Theorem 1 *In a simple behavioral game for channel closures where the Bridge plays first, the Nash equilibrium is "close channel normally" for the Bridge, and "punish" for the Gateway. Symmetrically, when the Gateway plays first, the Nash equilibrium is "close channel normally" for the Gateway, and "punish" for the Bridge.*

Proof. For the proof of Bridge starting the game first, we apply the backward induction. Thus, we observe that Gateway plays to punish the Bridge at the terminal node since $z' + y' > z'$ from Equation (2). Similarly, Bridge obtains a better payoff if it plays to close the channel normally because $y > 0$. Hence, it implies that the optimal solution for this game is that Bridge chooses *"close channel normally"* and Gateway chooses *"punish"*. That is the Bridge chooses to close the channel normally. In the event that Bridge chooses to broadcast a revoked state, Gateway plays to punish the Bridge. This equilibrium implies that the Bridge will always close the channel normally which is the intended behavior in LN.

In a similar fashion, when the Gateway starts the game first, we observe that Bridge plays to punish the Gateway as $y'' + z'' > y''$ from Equation (4) using backward induction. As expected, the Gateway becomes better off by playing to close the channel normally since $z > 0$. Therefore, the optimal solution for this game is that Gateway chooses *"close channel normally"* and Bridge chooses *"punish"*. Again, if the Gateway chooses to broadcast a revoked state, Bridge plays to punish the Gateway. Finally, we obtain that the Gateway will always close the channel normally which is again the intended behavior in LN. □

5.3 Security Analysis

In this section, we show how our proposed protocol addresses the attacks mentioned in Section 5.1.2.

5.3.1 Collusion Attacks

Broadcasting revoked states can be exploited by the Gateway and Bridge when they collude with each other to increase their chances of stealing money from the IoT and share that profit when they are successful. It is important to note that when the Gateway and Bridge are controlled by the same entity, this can cause IoT to lose funds. However, as explained in Section 5.2.2, the Gateway and Bridge are not controlled by the same entity since the Bridge is chosen by the IoT per our approach. Building on the analysis in Section 5.2.5, we formally analyze the collusion game as follows:

Theorem 2 *In a collusion game where the Bridge plays first, the Nash equilibrium is "close channel normally" for the Bridge and "punish", "do not share" for the*

Gateway. When the Gateway plays first, the Nash equilibrium is "close channel normally" for the Gateway and "punish", "do not share" for the Bridge.

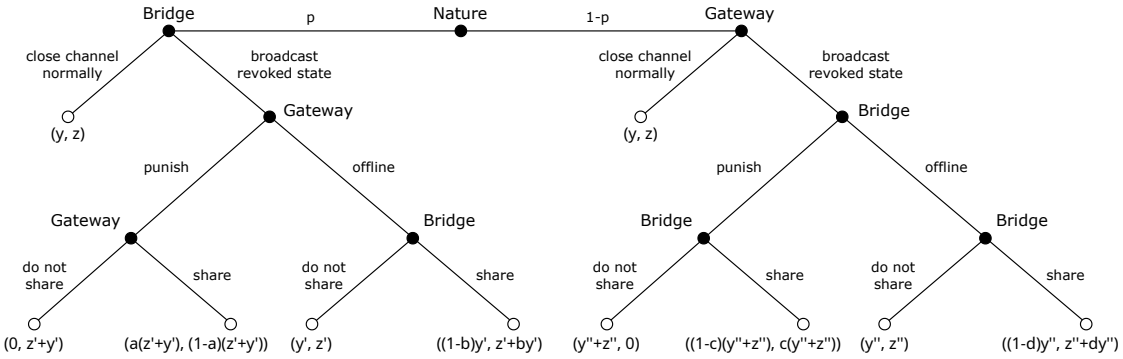

Figure 5.7: Extensive form of the collusion game between the LN gateway and the bridge LN node.

Proof. Similar to the proof of Theorem 1, we use the backward induction. The only difference from the previous proof is that, we include an additional subgame, i.e., the end subgame, in Fig. 5.7 for both cases. For the case of Bridge starting the game first, we apply the backward induction and check Gateway's and Bridge's payoffs at this end subgame (their second decision nodes). At its second decision node, Gateway chooses not to share profits since $z' + y' > (1 - a)(z' + y')$. Furthermore, Bridge also chooses not to share profits at its second decision node since $y' > (1-b)y'$. Here, a and b represent the ratios used to share the profits. After that, we continue the backward induction procedure, we observe that Gateway plays to punish the Bridge as $z' + y' > z'$ and Bridge chooses to close the channel normally because $y > 0$ as given in the proof of Theorem 1. Hence, the optimal solution for this game is that the Bridge chooses *"close channel normally"*, the Gateway chooses *"punish"* and *"do not share"*. In other words, the Bridge chooses to close the channel normally. In the even that Bridge chooses to broadcast a revoked state, the Gateway plays to punish the Bridge and not share the profits. This equilibrium implies that the

Bridge will not broadcast a revoked state as it results in Gateway getting all its funds and not share anything back. Thus, the Bridge will always close the channel normally, i.e., our protocol is secure against the collusion attacks.

For the case of the Gateway starting the game first, one should again apply the backward induction and check Bridge's payoffs at both end subgames. For both punish and offline cases, Bridge chooses not to share the profits since $y'' + z'' > (1-c)(y'' + z'')$ and $y'' > (1-d)y''$. Similar as before, c and d represent the ratios used to share the profits. Next, we observe that Bridge plays to punish the Gateway as $y''+z'' > y''$ from Equation (4) and Gateway chooses to close the channel normally because $z > 0$ which was also shown with the proof of Theorem 1. Similar to the first case, we then find that the optimal solution for this game is that the Gateway chooses *"close channel normally"*, the Bridge chooses *"punish"* and *"do not share"*. This implies that the Gateway chooses to close the channel normally. In the event that the Gateway chooses to broadcast a revoked state, the Bridge plays to punish the Gateway and not share the profits. The same reasoning about the equilibrium applies here and hence the Gateway will always close the channel normally, which again proves that our protocol is secure against the collusion attacks. □

5.3.2 Stealing IoT Device's Funds

Using (2,2)-threshold signatures for the LN operations secure the IoT device's funds in the channel since the LN gateway: 1) cannot send IoT device's funds in the channel to other LN nodes without generating proper HTLC and commitment signatures with the IoT device in a (2,2)-threshold signing; 2) cannot cause loss of IoT device's funds by broadcasting revoked states as shown in Section 5.3.1 and; 3) cannot cause loss of IoT device's funds by colluding with the bridge LN node again as shown in

Section 5.3.1 with the game theoretic security analysis. If we used LN's original signing mechanism, the LN gateway could move IoT device's funds in the channel without needing a signature from the IoT device. Consequently, the usage of (2,2)-threshold schemes along with the proposed modifications to LN's commitment transactions prevent IoT device from losing any funds.

5.3.3 Ransom Attacks

This is an attack where the LN gateway deviates from the protocol description. To put it with some examples, the LN gateway can say to the IoT device: "I will perform your channel closing request only if you pay me X amount of Bitcoins" or; "from now on, I will execute your payment sending requests only if you accept to pay an 10% increased service fee". This essentially turns into a game where the IoT device's best move is to reject the ransom attempt and just wait. Then, the LN gateway would just hold the IoT device's funds hostage for as long as it can in an attempt to deter the IoT device. This is a *deadlock* case where both parties just wait. It is clear that a *rational* LN gateway has no incentive to perform ransom attacks as it does benefit from these attacks assuming that the IoT device acts rationale. The best course of action for the LN gateway is to continue serving the IoT device and keep collecting the service fees. Thus, our proposed protocol protects the IoT device against ransom attacks.

5.4 Evaluation

This section describes the experiment setup to implement the proposed approach and presents the performance results.

5.4.1 Experiment Setup and Metrics

To evaluate the proposed protocol, we implemented it by modifying the source code of Core Lightning v0.9.3 [Blo23a] which is one of the implementations of the LN protocol written in C. The *hsmd* and *bitcoin* modules of Core Lightning were modified. The hsmd module manages the cryptographic operations and controls the funds in the channel. As the name suggests, the bitcoin module handles the Bitcoin script, signature and transaction routines. Specifically, we thresholdized the *funding_pubkey* and the associated Bitcoin private key as explained in Section 5.2.2. To the best of our knowledge, this is the first-ever work that implemented threshold cryptography for LN. Our implementation is publicly available in our GitHub repository at `https://github.com/startimeahmet/lightning`.

In the experiments, we used WiFi (IEEE 802.11n) and Bluetooth Low Energy (BLE) as the main communication protocol between the IoT device and the LN gateway. The purpose of performing the experiments with both WiFi and BLE is to investigate how different wireless technologies perform with our approach. Thus, it is essential for the rest of the components of the experiment setup to stay the same while changing the wireless communication method. In this direction, we created two experiment setups that we will call *WiFi setup* and *BLE setup*. For all experiments, we used Bitcoin's Testnet as the base-layer. Similar to Bitcoin's Mainnet, Testnet network consists of real Bitcoin nodes all around the world. However, Testnet Bitcoins do not have a monetary value unlike Mainnet which makes it more suitable for development and testing purposes.

Both setups use Raspberry Pi 4 Model B as the IoT device which is equipped with on-board dual-band IEEE 802.11b/g/n/ac wireless, Bluetooth 5.0 and BLE. For the LN gateway, we used a desktop computer with 2 Intel(R) Xeon(R) E5-2690 v4 CPUs and 32 GB of RAM. The desktop computer was connected to the Inter-

net through a Gigabit Ethernet connection inside Florida International University campus. It ran our modified version of the Core Lightning for the full LN node. For the full Bitcoin node, it ran *bitcoind* [The23] which is one of the most widely used implementations of the Bitcoin protocol. For the threshold operations, we used *Gotham city* [Zen23] which is a decentralized client/server Bitcoin wallet application that utilizes 2-party threshold ECDSA. Gotham city consists of *Gotham client* and *Gotham server*. Gotham client is the wallet application and was run on the desktop machine. Gotham server is a RESTful web service acting as a server for the threshold operations and was run on the Raspberry Pi. The other way is also possible (server on the desktop machine, client on the Pi) but it was easier to run the server on the Pi in our setup. Additionally, we created another LN node running the original Core Lightning v0.9.3 software in a DigitalOcean droplet in New Jersey to use in our experiments.

For the WiFi setup, we connected a TP-Link TL-WN722N 150Mpbs Wireless USB Adapter to the desktop machine. Using this adapter, we created a WiFi hotspot which the Raspberry Pi connected to. In this way, the Pi directly communicated with the desktop machine over the WiFi (IEEE 802.11n) connection.

For the BLE setup, we connected an Asus USB-BT500 Bluetooth 5.0 USB Adapter (supports BLE) to the desktop machine. Now, even though both the desktop machine and the Raspberry Pi have BLE connectivity, they cannot perform the threshold operations over BLE because the Gotham city is designed to work over TCP/IP. To overcome this issue, we enabled TCP/IP to work over BLE on the devices which is known as IP over BLE. A bunch of configuration had to be done for this to work such as installing *bluez* package, creating a Personal Area Network, configuring the master node and the slave node and more.

The further details of our experiment setups, additional technical details and the tools & scripts we used are provided in our GitHub page at `https://github.com/startimeahmet/LNGate2`.

To assess the performance of our protocol, we used the following metrics: 1) *Time* which refers to the communication and computational delays of the proposed protocol; 2) *Cost* which refers to the monetary costs associated with our proposed protocol; 3) *Bandwidth* which refers to the network usage of the IoT device and the minimum required bandwidth (data rate) for the IoT device for timely LN operations; 4) *Scalability* which refers to the scalability of the proposed protocol for increasing number of IoT devices and payments; 5) *Energy* which refers to the energy consumption of the IoT device when using the proposed protocol.

To compare our approach to a baseline, we considered the case where the LN gateway performs the LN operations by itself such as sending and receiving a payment. In other words, no IoT device is present, and all LN tasks are solely performed by the LN gateway. We will refer to it as *no IoT case* in the next sections.

5.4.2 Communication and Computational Delays

We first assessed the communication and computational delays of our proposed protocol. Computational overhead of running the protocol on the Raspberry Pi involves 3 different computations. These are the AES encryption of the protocol messages, HMAC calculations and (2,2)-threshold computations. We used Python's *pycrypto* library to encrypt the protocol messages with AES-256 encryption. The encrypted data size for the messages was 24 bytes. For the authentication of the messages, we used HMAC. To calculate the HMACs, *hmac module* in Python was used. Measuring the pure computation times of the (2,2)-threshold key generation

and signing is a little bit tricky because Gotham client and Gotham server are run on separate devices. We can instead run both the client and the server on the same device and use localhost for the server to eliminate any real network traffic. In this direction, we ran the client and the server on the Pi as well as the desktop PC to measure the best and worst cases of the pure computation times. We present these measured computation times in Table 5.1. All values are an average of 30 runs of the respective operation.

Table 5.1: Pure Computation Times

AES Encryption	**HMAC Calculation**	**(2,2)-Threshold Key Generation**		**(2,2)-Threshold Signing**	
		IoT	**Desktop**	**IoT**	**Desktop**
15 ms	< 1 ms	1.78 s	0.53 s	166 ms	74 ms

As can be seen from the results, the overhead of the AES encryption and HMAC calculation are negligible. (2,2)-threshold key generation takes 1.78 seconds on the Pi and 0.53 seconds on the desktop machine. Thus, the real delay is between these two values and it is not critical for LN operations since it is done 1-time at channel opening as explained in Section 5.2.2. The (2,2)-threshold signing on the other hand takes 166 ms on the Pi and 74 ms on the desktop machine which is much quicker than key generation.

We then measured the execution time of our protocol for 5 different LN operations using WiFi and BLE as the communication method between the Pi and the desktop machine. The 5 LN operations we considered are: 1) Channel opening, 2) Channel closing, 3) Sending an invoice payment, 4) Sending a key send payment, 5) Receiving a key send payment. Execution time of an LN operation can be broken down to the sole execution time of the LN operation at the LN gateway plus the execution time of the (2,2)-threshold operations between the IoT device and the LN gateway. The results are presented in Table 5.2 and 5.3 and they are used in

other experiments to evaluate the timeliness of the protocol. Note that, the channel opening and channel closing delays in Table 5.2 do not include the confirmation times on the blockchain.

Table 5.2: Execution Times of Channel Opening and Closing for "WiFi", "BLE" and "No IoT" Cases

	from	**to**	**WiFi Delay**	**BLE Delay**	**No IoT Delay**
Channel Opening	LNGate^2	VIOLETWALK	1.73 s	2.85 s	0.32 s
Channel Closing	LNGate^2	VIOLETWALK	0.91 s	1.35 s	0.18 s

Table 5.3: Execution Times of Payment Operations for "WiFi", "BLE" and "No IoT" Cases

	from	**to**	**WiFi Delay**	**BLE Delay**	**No IoT Delay**
Sending Invoice Payment	LNGate^2	VIOLETWALK	1.07 s	2.75 s	0.31 s
Sending Key Send Payment	LNGate^2	VIOLETWALK	1.01 s	2.97 s	0.33 s
Receiving Key Send Payment	VIOLETWALK	LNGate^2	1.08 s	2.93 s	0.3 s

The *'from'* and *'to'* fields in the Table 5.2 and 5.3 are the LN node aliases. For example, the second row of Table 5.3 should be interpreted as follows: A key send payment was sent from the node with alias *LNGate*^{2} to the node with alias *VIOLETWALK*. *LNGate*^{2} is our node with the threshold modifications running on the desktop machine that is located in Florida. *VIOLETWALK* is our node running the original LN software on a DigitalOcean droplet in New Jersey. Each delay value in Table 5.2 and 5.3 is an average of 30 executions of the respective LN operation for statistical significance.

We first present the channel opening and closing delays in Table 5.2 as they are 1-time operations and their execution times are not critical since they should be done in advance before the payment services are used. Therefore, blockchain confirmation delay of these operations also will not have impact on the actual payment transactions. It is important to also note here that the users can use any of their existing channels to make payments which alleviates the need to open a specific channel for every transaction consequently the need to wait for channel openings.

As can be seen in Table 5.2, the gap is not significant compared to no IoT case. Specifically, WiFi case has around 1.4 seconds of extra delay compared to the no IoT case which mostly comes from the computation time of the threshold key generation. Channel opening delay of the BLE case is around 1 second longer compared to the WiFi case which is normal due to BLE's limited bandwidth. For the channel closing, WiFi case is only 0.7 seconds slower than the no IoT case which is again mostly due to the computation time of the threshold signing operations. A mutual channel close involves several rounds of threshold signing for peers to agree on the closing fee as explained in Section 5.2.5. BLE is slower by approximately 0.4 seconds than the WiFi due to its lower bandwidth.

Next, we show the results of payment sending and receiving in Table 5.3 which are of more importance to the usability of the protocol. As can be seen, regardless of the payment operation, the WiFi case took around 1 second while no IoT case took 0.3 seconds and BLE case took close to 3 seconds. To interpret these results, we need to look at how many times threshold signing is performed in an LN payment. As shown in Section 5.2.3 and 5.2.4, each HTLC requires two threshold signing. In our experiments, we realized that sometimes a single payment creates multiple HTLCs resulting in performing more than two threshold signing. Specifically, invoice and key send payments in this experiment had four threshold signing operations which took between (296, 664) ms. This tells us that the extra 0.7 seconds delay in the WiFi case mostly came from the computations of the threshold signing operations. The communication delays constitute a very small part of WiFi's overall delay.

We can also see that BLE adds roughly another 1.8 seconds to the delay compared to WiFi due to its limited data rates for transmitting threshold messages. Nevertheless, a delay of 2.97 seconds to send a key send payment under BLE can be considered fast enough for most scenarios. In particular, WiFi and BLE delays are

comparable to or even less than that of any typical credit card payment delay which can take two to three seconds to get approved. This time increases further with two factor authentication services for security (e.g., more than 10 seconds) [RSD+19].

5.4.3 Cost Analysis

We consider and analyze the following costs associated with our proposed protocol: 1) The on-chain fees for channel opening and closing; 2) The fees that are charged by the LN gateway for IoT device's payments; 3) Forwarding fees that are charged by the nodes on a payment route (i.e., bridge LN node).

According to our experiment results; a channel opening transaction signed with (2,2)-threshold ECDSA, incurs a fee of only 222 satoshi when it is desired for the transaction to be included in the next block[3]. Similarly, a mutual channel closing transaction signed with (2,2)-threshold ECDSA, costs only 183 satoshi[4]. At current Bitcoin price of $30,000, these fees correspond to *6.6 cents* and *5.5 cents* respectively which are basically negligible considering these operations are 1-time.

LN gateway's service fee for payments entirely depends on the LN gateway's choice. Forwarding fees, again entirely depend on nodes' choices that are on a payment path. In LN, nodes charge a fixed fee each time they route a payment which is called the *base fee* [Bit19]. There is also *fee per satoshi* that the nodes charge proportional to the satoshi value of the payments they route [Bit19]. To measure an approximate value for the forwarding fee, we sent 30 LN payments

[3]See this channel opening transaction which is signed using (2,2)-threshold ECDSA: `https://blockstream.info/testnet/tx/f2dfec159f66be6b785c97b247c44d6efd6fc9cd40a1b2d800386cec450f797a`

[4]See this mutual channel closing transaction signed with (2,2)-threshold ECDSA: `https://blockstream.info/testnet/tx/9714d8796425b472ccfa8e049b83f72d59d1505cc805d9a3588fdc8b6865213d`

using different routes for each. The average of the forwarding fees was *2 satoshi* which is again negligible.

Most importantly; the channel opening & closing fees and forwarding fees do not change with the introduction of (2,2)-threshold ECDSA. These values we measured are exactly the same for a regular LN node running the original LN software. Therefore, our protocol does not bring extra cost overhead for the LN operations.

5.4.4 Network Usage and Bandwidth Analysis

In this experiment, we investigate: 1) the *network usage* of the IoT device; 2) the *minimum required bandwidth* for the IoT device to timely complete the LN operations with the LN gateway. Our motivation for finding the minimum required bandwidth is to understand if very low bandwidth IoT devices can participate in our protocol. To measure the network usage, we used Wireshark packet analyzer software. We captured the packets on a specific network interface, e.g., `wlxec086b180e1b` for WiFi, `pan0` for BLE. After the capture, using the *endpoint statistics* of Wireshark, we collected the number of packets and bytes information. We considered the same 5 LN operations which are: 1) Channel opening, 2) Channel closing, 3) Sending an invoice payment, 4) Sending a key send payment, 5) Receiving a key send payment. All LN operations were performed 5 times and the average values were calculated.

The network usage of WiFi and BLE cases were the same and it ranged between ($\approx$ 16,000, $\approx$ 42,000) bytes where ($\approx$ 80, $\approx$ 200) network packets were exchanged between the IoT device and the LN gateway. Channel opening and closing were less intensive in terms of network usage compared to sending and receiving payments.

Now, the minimum required bandwidth for the IoT device can be calculated. If we focus on sending key send payments where the IoT device have to exchange

around 42,000 bytes in 2.97 seconds (Table 5.3), then the minimum required bandwidth is 113 kbps. This is a reasonable bandwidth requirement considering that even the low-power wireless technology 6LoWPAN supports a data rate of 250 kbps [HT11].

5.4.5 Scalability Analysis

To test the scalability of our approach, we performed additional experiments using increased number of Raspberry Pis instead of just 1. This setup consists of 5 Pis all associated with a separate LN node running on the desktop computer. Basically, the desktop machine ran 5 LN nodes and 5 Gotham clients to serve each Pi. On the other hand, each Pi ran a Gotham server to perform the LN operations with their corresponding LN nodes on the desktop machine.

To test the scalability of our approach, we conducted two different experiments: 1) Changed the number of sender IoT devices (i.e., Pis) where each of them is sending 100 payments at once to a single recipient LN node. The goal of this experiment is to investigate how increasing the number of Pis will impact the computational overhead on the LN gateway as it will now serve more Pis at the same time. In particular, we are interested to see how it will affect the overall delays; and 2) Changed the number of payments from a single sender IoT device to a single recipient LN node. The goal here is to see how increased number of payments impact the performance by keeping the computational overhead on the LN gateway constant by only serving one IoT device. For this experiment case, we also used an unmodified LN node as the sender node to create a baseline case. This way, we can compare the results with our approach to see if any change in the delays are due to LN's internal mechanisms or related to our approach. For each of these experiment cases, we created bash scripts

to automate the process. Each script was run 30 times and average results were used for statistical significance. For the recipient LN node, we used our *VIOLETWALK* node.

Table 5.4: Effect of Increasing the Number of IoT Devices on Payment Delays

Number of IoT Devices	**Average Payment Delay**
1	3.50 seconds
2	3.84 seconds
3	4.98 seconds
4	5.43 seconds
5	7.18 seconds

Overhead of increased number of IoT devices: The results of this experiment are presented in Table 5.4. As can be seen, increasing the number of IoT devices that the LN gateway is serving resulted in an increase on the overall payment delays. Each Pi generated 100 key send payments all at once resulting in a burst of payments that need to be processed by the LN nodes running at the LN gateway. Thus, the increase in delays is not unusual and quite expected as the LN gateway communicated with each Pi more frequently for threshold signing operations. Additionally, the increase is not linear. When we check the results in Table 5.5, we observe that there is already a natural increase in delays when an LN node receives increasing number of transactions. This is because the LN protocol have to process all these HTLCs simultaneously which causes payment settling times to increase. Thus, we can understand that most of the increase in delays in Table 5.4 are due to the LN overhead for processing the payments. The additional overhead coming from using our approach is minimal, which demonstrates that our approach can scale. Nevertheless, if certain delay requirements are to be met, additional LN gateways can be deployed to reduce the load depending on the number of IoT devices to be deployed.

Overhead of increased number of payments: The results of this experiments are shown in Table 5.5. Increasing the number of key send payments from a single sender resulted in increased delays for both threshold and baseline approaches. This clearly tells us that when an LN node processes more payments, some of the payments settle after the others resulting in a greater average delay. Thus, the increase in the delay is not relevant to our approach rather related to the LN's internal mechanisms. However, the threshold delays are still slightly higher than the baseline delays as expected which is due to the time spent on performing the threshold signing operations with the Pi.

Table 5.5: Effect of Increasing the Number of Transactions on Payment Delays

Number of Tx	**Threshold Delay**	**Baseline Delay**
100	3.50 seconds	2.65 seconds
200	5.87 seconds	4.59 seconds
300	8.65 seconds	6.96 seconds
400	10.53 seconds	8.96 seconds

5.4.6 Energy Consumption

As many IoT devices are powered with batteries, it is important to also analyze their energy consumption. In this direction, we used a MakerHawk USB multimeter device which can accurately measure the power consumption of a Raspberry Pi in real-time. A photo of this setup is shown in Fig. 5.8. We report the energy consumption of the Pi when it is idle and sending key send payments. We considered two ends of the spectrum: 100 and 400 transactions to see the difference. For the idle cases, we measured the energy consumption for the duration of time spent on sending the transactions. The results are shown in Table 5.6.

According to these results, a single transaction only costs around 5 μWh which indicates that the additional energy overhead that comes with our approach is mini-

Figure 5.8: Our energy consumption setup for the Raspberry Pi using the MakerHawk USB Multimeter.

Table 5.6: Energy Consumption of the IoT Device

Number of Tx	Sending Payments	Idle
100	3.56 mWh	3.03 mWh
400	11.76 mWh	10.13 mWh

mal. Additionally, the overhead remains constant even if the number of transactions increase. Using battery solutions such as PiSugar[5], Pis can run on battery around 10 hours non-stop. However, in our approach, we do not need the Pis to be running all the time. They can power-on only when they need to perform the threshold operations with the LN gateway and thus, can run on battery for days.

5.4.7 Comparison with Other Methods

Since there are not any reliable comparison options, we chose to implement the state-of-the-art method for regular online payments which is the credit card payments.

[5] `https://www.pisugar.com/`

Specifically, we used Square's Web Payments SDK[6] to implement the standard credit card payments. A card payment utilizes 3D Secure (3DS) also known as Strong Customer Authentication (SCA). Square lets developers test credit card payments using their Sandbox servers. Thus, we installed this implementation into one of the Raspberry Pis and measured the delay to make card payments to their server. We tested the delays both with 3DS and without and present these results in Table 5.7. The results clearly show that LN payments with our approach is much quicker than credit card payments even when 3DS is not enabled.

Table 5.7: Comparison with Regular Credit Card Payments

Method	**Payment Sending Time**
Credit Card Payment with SCA	$\approx$ 5 s
Credit Card Payment w/o SCA	$\approx$ 3 s
LNGate[2]	1.01 s

5.5 Limitations

Our proposed protocol requires changes to the LN protocol. Specifically, we propose changes to LN's BOLT #2 and commitment transactions. This will require ensuring that the updated LN software should be run on the LN gateway and bridge LN nodes in order to be able to support the IoT device transactions.

Additionally, our protocol requires the IoT device to be online for receiving payments. In an ideal setting, the IoT device should be able to receive a payment even when it is offline. It is important to note that this is also a problem for LN in general [Kuw19].

[6] `https://github.com/square/web-payments-quickstart`

CHAPTER 6

LNMESH: WHO SAID YOU NEED INTERNET TO SEND BITCOIN? OFFLINE LIGHTNING NETWORK PAYMENTS USING COMMUNITY WIRELESS MESH NETWORKS

Bitcoin is undoubtedly a great alternative to today's existing digital payment systems. Even though Bitcoin's scalability has been debated for a long time, we see that it is no longer a concern thanks to its layer-2 solution Lightning Network (LN). LN has been growing non-stop since its creation and enabled fast, cheap, anonymous, censorship-resistant Bitcoin transactions. However, as known, LN nodes need an active Internet connection to operate securely which may not be always possible. For example, in the aftermath of natural disasters or power outages, users may not have Internet access for a while. Thus, in this chapter, we propose LNMesh which enables offline LN payments on top of wireless mesh networks. Users of a neighborhood or a community can establish a wireless mesh network to use it as an infrastructure to enable offline LN payments when they do not have any Internet connection. As such, we first present proof-of-concept implementations where we successfully perform offline LN payments utilizing Bluetooth Low Energy and WiFi. For larger networks with more users where users can also move around, channel assignments in the network need to be made strategically and thus, we propose 1) minimum connected dominating set; and 2) uniform spanning tree based channel assignment approaches. Finally, to test these approaches, we implemented a simulator in Python along with the support of BonnMotion mobility tool. We then extensively tested the performance metrics of large-scale realistic offline LN payments on mobile wireless mesh networks. Our simulation results show that, success rates up to 95% are achievable with the proposed channel assignment approaches when channels have enough liquidity.

6.1 Feasibility Study

To demonstrate the feasibility of the approach, we first created a proof of concept implementation. This section provides the details of this implementation and the corresponding results. The proof of concept will shed light on how to design such a system in large-scale which will be tackled in the next sections.

6.1.1 Implementation Environment

Our implementation utilizes 8 Raspberry Pi 4 Model B each having a 64GB SD card loaded with 64-bit Raspberry Pi OS (Raspbian). Each of these Raspberry Pis support Bluetooth Low Energy (BLE) and WiFi interfaces with dual-band IEEE 802.11b/g/n/ac and Bluetooth 5.0. For the Bitcoin nodes, we installed *bitcoind v0.23.0* [The23] and for the LN nodes, we used *Core Lightning v0.11.2* [Blo23a]. All the transactions were performed on Bitcoin's Testnet. To fund the LN nodes with Testnet Bitcoins, we used a faucet[1]. Note that in LN, channels need to be opened in advance and transactions can be sent to recipients over multi-hop routes. If there is no route to the recipient, the payment will not go through.

6.1.2 Bluetooth Implementation and Results

We first tested the offline LN payments on a BLE setup to have an initial proof of concept. The setup consists of 8 Pis where one of Pis is the master and the rest are slaves. LN works with TCP/IP thus BLE alone cannot be used to perform LN operations between the nodes. BLE setup needs to be adjusted such that the TCP/IP can work over BLE. To do this, we did some configuration on the Pis such as installing *bluez-tools*, creating a personal area network, configuring the master

[1] `https://coinfaucet.eu/en/btc-testnet/`

node and slave nodes and more. However, doing only that is not enough as we are also trying to create a mesh network. Such a setup is then called IP mesh over BLE with no known implementation to-date. As we are only trying to test the feasibility of the concept, we instead created a *IP-over-BLE Star Network*. Basically, one of the Pis were designated as the master and other 7 connected to it as slaves. The full details and step-by-step guide of this implementation is given at our GitHub page at `https://github.com/startimeahmet/LNMesh/tree/main/BLE_star`.

Once this IP-over-BLE star network setup was ready, the next step was to open the LN channels. For this, we again chose a star topology where all slaves opened a channel to the master. The physical placement of the slaves around the master was arbitrary. The virtual topology of this setup is illustrated in Fig. 6.1.

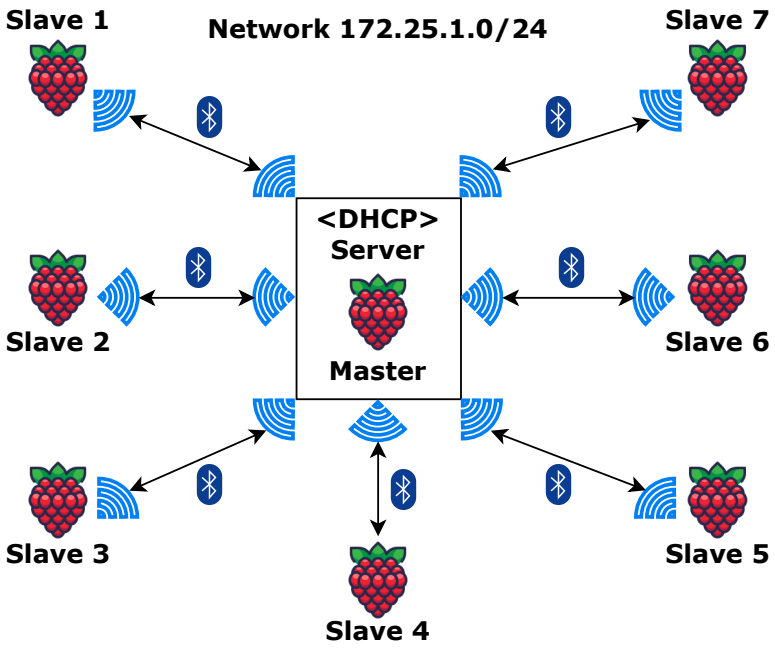

Figure 6.1: Topology of the IP-over-BLE star network.

Once the Bitcoin and LN nodes of all Pis were synced, we cut the Internet connection on all the Pis at the same time to take them all offline. This is a critical part in the experiments as our motivation is to test offline LN payments. In our

tests, we realized that, cutting the Internet connection puts the Bitcoin and LN nodes into *searching for peers* mode. Both nodes still keep running but lose their existing peers because of having no Internet connection. Thus, we just went ahead and connected each LN node on the slave Pis to the LN node on the master Pi using their local IP addresses `172.25.1.xxx` with the command `lightning-cli connect pubkey@IP:port`. In a real deployment, this process can be automated with a script which will automatically get the local IPs of the peers and force a reconnect. This way, we did not need to make any modifications on the LN or Bitcoin software to make offline LN payments work.

For the experiments, we executed 100 payments in total where we randomly selected two nodes among the 8 nodes to send payments to each other. This was automated with a bash script we wrote. Since the LN channels were opened to form a star topology, payments can either be between the slaves and the master (direct payments) or between the slaves (1-hop payments). All executed payments were successful. The average delay was around 1.5 seconds for direct payments and around 2.5 seconds for 1-hop payments.

6.1.3 WiFi Implementation and Results

Next, we tested the concept with WiFi as it will offer longer coverage. Unlike the BLE case, we created a full mesh network (i.e., not through a master) for the WiFi experiments. Thus, it involves a more complicated setup since the nodes have to be placed at a distance from each other to create a mesh topology. To create the WiFi mesh network, we used *batman-adv* [Fre23] which is a routing protocol specifically designed for mobile ad-hoc networks and is part of the Linux kernel. One of the Pis were setup as the mesh gateway to provide Internet connectivity to

the rest of the Pis in the mesh. For that, we used an additional generic Wireless USB adapter on the gateway to connect it to the Internet when necessary. This Internet connectivity was only used to sync the Bitcoin and LN nodes on the Pis. The mesh network was created inside our university building and the nodes do not necessarily have line of sight propagation between each other. The full details and step-by-step guide of this implementation is given at our GitHub page at `https://github.com/startimeahmet/LNMesh/tree/main/WIFI_mesh`.

In Fig. 6.2, we show the topology of the nodes inside FIU Engineering Center building and their connections to each other on the mesh level. The rectangles in the figure represent the rooms and walls. The dashed lines on the figure represent the neighbors of each node on the mesh network which were identified with the command `sudo batctl n` where *batctl* is the configuration and debugging tool for *batman-adv*.

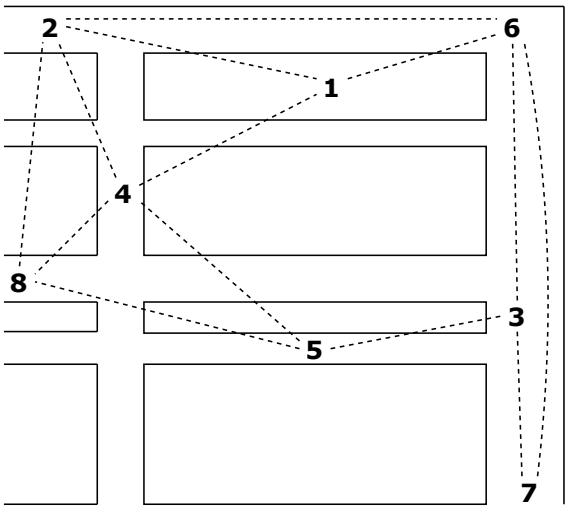

Figure 6.2: Placement of the Pis in the building and the resulting mesh topology for the WiFi mesh setup. Dashed lines show the mesh links.

Next, we created several LN topologies by opening channels between the nodes. These topologies can be seen in Fig. 6.3. First topology is the same as the mesh topology where each node has a channel with its neighboring nodes. Second one is a

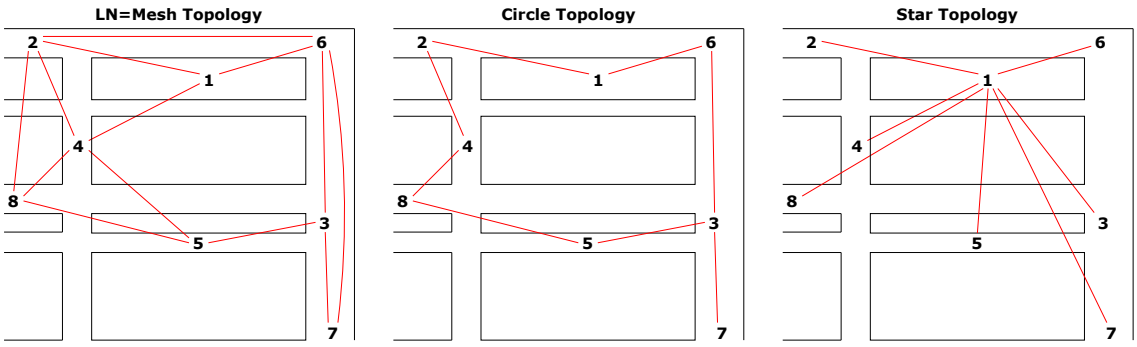

Figure 6.3: Illustrations of different LN topologies tested on the WiFi mesh setup. Red lines show the LN channels.

circular topology where the channels form a complete loop. Finally, the third one is a star topology where all nodes have a channel to the gateway node. Same as the BLE case, we cut the Internet connection of the nodes after the channels were opened and peered them with each other on LN using their local IPs `192.168.199.xxx`.

For all these topologies, again similar to the BLE case, we performed 100 payments between two randomly selected nodes among the 8 nodes we had. Payment amounts were set to 5,000 satoshi and all channel capacities were initially set to 100,000 satoshi. The procedure of choosing the nodes randomly and executing the payments between the nodes was automated using a bash script. Randomization was done using fixed seed values to prevent different nodes to be selected for different experiment runs. The results of these experiments are presented in Table 6.1.

Table 6.1: WiFi Mesh Experiment Results

Topology	**Success Rate**	**Average Delay**	**Total Number of Channels**
LN=Mesh	95%	5.95 sec	13
Circle	53%	6.31 sec	8
Star	60%	4.76 sec	7

Highest success rate was observed with the LN=Mesh topology which makes sense as it has more channels than the other two thus more routing options/less

failures. Circle topology on the other hand has the lowest success rate and the most delay which again makes sense because routing options are limited and payments have to go through more hops. Finally, we see that the star topology performs slightly better than the circle with the least delay among all options which is logical as the payments are either direct or 1-hop payments. In general, these results show us that while offline payments are possible, the way the channels are assigned makes a major impact on the success rate when considering mesh topologies. Therefore, we tackle this problem next.

6.2 Channel Assignment Approaches

In the previous section, we showed that offline LN payments can indeed be realized in a wireless mesh network utilizing WiFi or BLE. Now in this section, we will present techniques to extend such a setup to a large number of users and channels, which can be used in a real-life application within a community when they all lose Internet connection at the same time due to a disaster such as a hurricane/earthquake etc. In this direction, we first describe how to handle large-scale mesh topologies that change over time (i.e., mobile) and then offer two different channel assignment techniques for mobile community wireless mesh networks.

6.2.1 Problem Motivation and Handling Mobility

Opening and closing LN channels require Internet connection since they are on-chain Bitcoin transactions broadcast to the Bitcoin network. This means that the users have to open their channels before they go offline. Therefore, our problem boils down to the following: Given a community of people moving around in a neighborhood during a day, how to decide who opens a channel to who so that the overall success

rate of the payments in the network will be satisfactory when people do not have access to the Internet?

A good starting point for deciding how to open channels in a wireless mesh network is by looking at the individual mesh connections of the users. We could open an LN channel for every mesh link and have an LN topology that is the same as the mesh topology. However, this will not be possible since we are dealing with a mobile topology that changes over time. Additionally, opening a channel for every single mesh link would be very costly as it would cause too many channels to be opened that would require higher initial investment from all the users. Instead, a more optimized way is to look at users' mobility patterns and make channel assignments based on these patterns to reduce the number of channels opened. Mobility pattern of a user can be interpreted as how frequently the user move around other users and how often it gets close to others. In real life, this corresponds to how often people interact with each other or with stores, gas stations and markets around them.

Algorithm 3: Mobility-Aware Mesh Topology Generation

Input : Mesh distances *data* with following 4 columns: [*source*, *target*, *time*, *distance*]

Output: Mobility-aware mesh topology G_{mesh}

```
1  define k;    /* the metric to include the nodes in the new topology */
2  define d;                                  // wireless coverage of the nodes
3  initialize empty G_mesh;
4  for each (source, target) in data do
5  |   count = 0;            /* how many times two users get close enough */
6  |   for each time in data do
7  |   |   if distance ≤ d then count += 1;
8  |   end
9  |   if count ≥ k then
10 |   |   G_mesh.add_edge(source, target);
11 |   end
12 end
13 return G_mesh;
```

To model these mobility patterns, we need to analyze the mobility behavior of the users for a period of time within a specific neighborhood. For instance, if we can collect mobility traces for certain users during a day in a neighborhood, these traces can then be merged to create instant mesh topologies at certain times. In other words, we can get a snapshot of mesh topologies at certain intervals and then identify the most probable neighbors of each user in average. This information can be combined to create an average wireless mesh topology of users during a typical day. This topology will give us a hint on how to assign channels. We came up with Algorithm 3 to create this topology.

Algorithm 3 is pretty straightforward in the sense that it gets as input the distance information between all possible users in the mesh at different times of the day and outputs the *mobility-aware mesh topology*. For all given times (line 6), it checks the number of occurrences any two users *(source, target)* (line 4) get close to each other less than the distance d (line 7). Here, d stands for the wireless transmission range of the users. If the number of occurrences is at least k times (line 9), we decide that these two users frequently interact with each other and add them to the mobility-aware mesh topology (line 10).

6.2.2 Problem Definition and Formulation

Now that we have the mobility-aware mesh topology which is fixed (i.e., does not change with time anymore), the next problem is how to assign the LN channels on it. The problem of channel assignment for mobile ad-hoc networks can be modeled based on one of the variants of the well-known transportation problem [App73]. Mainly, the goal in this problem is to move certain supplies from suppliers to warehouses through certain routes that come with specific costs. In our case, the supplies

would correspond to payments, suppliers and warehouses would be users and the cost of the route would be associated with the capacity (i.e., higher capacity channels are cheaper while lower capacity ones are expensive). However, in our case, there is an additional problem since we are trying to prune the resultant topology to reduce the number of channels opened. This problem is related to the connectivity of the topology. From a given topology which represents an undirected graph $G\ (V, E)$ where V is the set of users and E are the wireless links among them, we need to guarantee that the selected links eventually form a connected topology so that any payment can reach any destination. This connectivity constraint makes the problem intractable for sensor networks as shown in [SSSA11]. Therefore, we opt to follow approaches which can offer connected topologies with minimal number of links. Specifically, we will prune the topologies created by Algorithm 3.

To this end, one easy solution is to utilize *spanning trees* from graph theory which has been widely used in the context of mobile ad-hoc networks for various purposes [KSAU21], [BFG$^+$03] since they guarantee connectivity with minimum number of links. In graph theory, spanning tree of a given undirected graph G is a subgraph of G that includes all the vertices of G with minimum number of edges. That means, if G has n vertices, a spanning tree of G has $n - 1$ edges.

Finding the minimum spanning tree (MST) has been well studied, and there are polynomial time algorithms such as Kruskal's or Prim's [WC04] to find the MST if each link has an associated cost (i.e., weight). In our case, the different link costs used for an MST are not applicable since once created, each link is equal in terms of serving as an LN channel. In other words, each link will have the same cost. Therefore, any spanning tree would be applicable to our solution. Nevertheless, we recognize that even if the number of links (and thus the number of LN channels to be opened) will be same for any spanning tree topology, the degree of nodes (i.e.,

the number of neighbors for a node) may differ significantly. In other words, the variance of the node degrees may change from topology to topology.

Therefore, we plan to pursue two approaches to form our spanning trees: Using minimum connected dominating set (MCDS) [GK98] and uniform spanning tree (UST) [PW98] concepts. Next two sections are dedicated to these approaches.

Algorithm 4: MCDS-based LN Topology Generation

Input : Mobility-aware mesh topology G_{mesh}
Output: MCDS-based LN topology $G_{LN_{MCDS}}$

```
1  G_MCDS = compute_MCDS(G_mesh);     /* compute the MCDS graph using any
     known MCDS algorithm */
2  G_MCDS_nocycles = compute_MST(G_MCDS);   /* compute MST of MCDS graph to
     remove possible cycles */
3  G_LN_MCDS = G_MCDS_nocycles;                  /* initialize LN topology */
4  for each node in G_mesh do
5      if node not in G_MCDS then
6          neighbors = G_mesh.neighbors(node);
7          possibilities = neighbors ∩ G_MCDS;
8          G_LN_MCDS.add_edge(node, possibilities[0]);        /* choose the first
             possibility */
9      end
10 end
11 return G_LN_MCDS;
```

6.2.3 Minimum Connected Dominating Set Approach

As mentioned before, we are trying to assign minimum number of channels in the network to reduce the burden on the users to open many channels which requires monetary investment. In this direction, we can first find the core vertices of the mobility-aware mesh topology graph and then follow this core to assign channels. The rationale behind this is to form a topology such that any node can reach any other node through this core with relatively shorter paths. This can indeed be measured through metrics such as *closeness centrality* and *betweenness centrality* [Gol13] from graph theory. Shorter paths increase the success rates because there

are less channels on the payment path that might cause the payment to fail. The other motivation behind determining a core is to allow any potential businesses to be part of the core if they offer a wireless node to interact with the users.

Core vertices in a topology can be found with a minimum connected dominating set (MCDS) algorithm. Connected dominating set D (i.e., dominator nodes) of a graph G has the following properties:

- D forms a connected subgraph of G.
- If a vertex in G is not in D (i.e., a dominatee node), it is adjacent to a vertex in D.

Thus, MCDS finds a D with minimum number of vertices. We present the pseudocode of our MCDS-based LN topology generation algorithm in Algorithm 4.

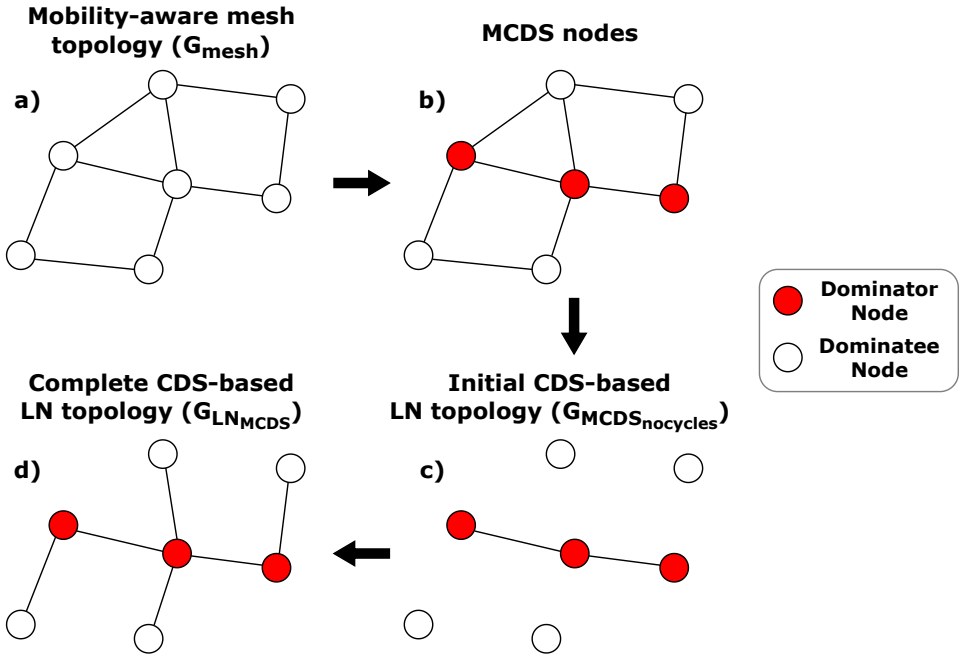

Figure 6.4: An illustration of the workflow of Algorithm 4

Algorithm 4 takes the mobility-aware mesh topology graph as an input and returns a processed version with fewer number of links. We first compute the MCDS

graph of the mobility-aware mesh topology graph (line 1). For finding the MCDS, any known algorithm in the literature can be used [GK98]. However, when we find it, we remove any cycles within this MCDS by computing its MST (line 2). This resulting graph is also the starting point to create our LN topology thus we initialize the LN topology graph to the MCDS graph with no cycles (line 3). The idea is to enlarge the MCDS graph with no cycles until each dominatee node has a channel to one dominator node. To do this, we iterate over each node in the mobility-aware mesh topology graph that is not a member of the MCDS (i.e., dominatee nodes) and get their neighbor set (line 4-6). In this neighbor set, we check for the nodes that are in the MCDS (line 7). These nodes are potential candidates to open a channel to because they are already in the MCDS and have a link to the dominatee node in the mobility-aware mesh topology. We take the first candidate and add this (dominatee, dominator) pair to the LN topology graph (line 8). In this way, every node will be connected and there will not be any cycles, essentially another spanning tree. Once this process is done for all the dominatees, we output the LN topology graph. The way this algorithm works is also illustrated in Fig. 6.4.

6.2.4 Uniform Spanning Tree Approach

As an alternative to finding the MCDS, we can calculate spanning trees from the mobility-aware mesh topology graph using another method. As mentioned, since a graph can have multiple spanning trees, we can randomly select a spanning tree among all possible spanning trees of G with equal probability. This is referred to as Uniform Spanning Tree (UST). USTs can be found using Wilson's algorithm which uses loop-erased random walks to generate them [PW98]. We show an example UST in Fig. 6.5 of the same graph in 6.4. The rationale behind this choice is to be able

to compare random selection to a deliberate one (i.e., MCDS-based) and investigate the impact through experimentation.

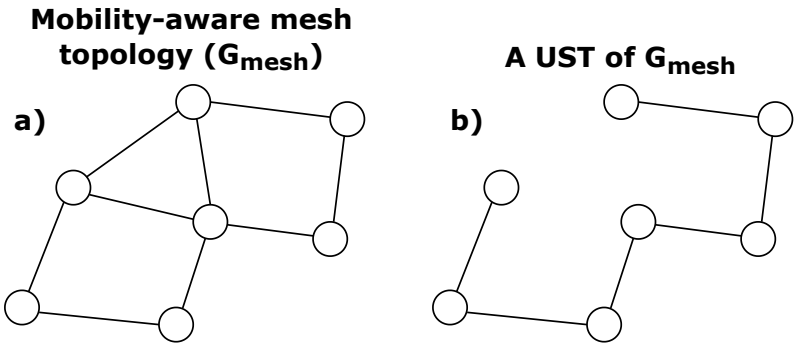

Figure 6.5: An example UST generated from G_{mesh}

6.3 Simulations

To test our proposed approaches explained in Section 6.2, we implemented a simulator in Python. It mostly utilizes the *networkx* and *pandas* libraries in Python to create the graphs of the mesh and LN topologies and perform operations on them. The full source code of our simulator is available in our GitHub page at `https://github.com/startimeahmet/LNMesh/tree/main/simulator`.

6.3.1 Implementing Mobility

Before starting the simulations, we need to generate realistic mobility data to be able to create our mesh network. For this, we used the *BonnMotion* software which is a mobility scenario generation and analysis tool written in Java [AEGPS10]. Using this software, we can generate mobility scenarios based on popular mobility models within a given community. An example scenario generated by BonnMotion is shown in Table 6.2. As can be seen, a scenario file created by BonnMotion has 4 columns

in it which are [*node, time, x, y*]. For example, row 694 in Table 6.2 is interpreted as follows: Node 1 was at coordinates (260.0223, 453.6815) at time 79.8913.

Table 6.2: An example scenario created by BonnMotion for the duration of 21,600 seconds (6 hours) for 100 nodes

	node	**time**	**x**	**y**
row 1	0	0	640.9129	574.3036
row 2	0	18.1457	618.9429	574.0308
row 3	0	21.9939	618.9429	568.2012
.	.	.	.	.
.	.	.	.	.
row 692	0	21592.6878	302.5685	597.1364
row 693	1	0	268.3588	329.3055
row 694	1	79.8913	260.0223	453.6815
row 695	1	275.8172	464.9036	457.9410
.	.	.	.	.
.	.	.	.	.
row 1320	1	21600	474.0779	697.9201
.	.	.	.	.
.	.	.	.	.
row 70281	99	0	327.1091	598.0677
row 70282	99	141.9900	464.2400	603.2721
row 70283	99	198.8276	472.7828	505.9701
.	.	.	.	.
.	.	.	.	.
row 70859	99	21600	255.2838	574.6977

In short, a scenario includes nodes' coordinates at different times for the duration of the scenario. Thus, a scenario is essentially a network topology that changes over a period of time. We can also input real maps to the software to create more realistic scenarios. In this direction, we first chose a specific neighborhood around our school, Florida International University (FIU). Then, we extracted the map of FIU's Engineering Center Campus and its surroundings using the Java OpenStreetMap Tool (JOSM)[2]. The map is shown in Fig. 6.6. The bounding box of our

[2] `https://josm.openstreetmap.de/`

map in longitude/latitude format is [*-80.3733, 25.7647, -80.3653, 25.7722*]. With this map, we generated a number of mobility scenarios on the BonnMotion software using the Map-based Self-Similar Least-Action Walk (MSLAW) [SA13] algorithm. MSLAW was a suitable option for our case since it best captures moving people in a neighborhood and also it can take real maps as inputs. The scenario duration was 6 hours, the min and max speed of the pedestrians were 0.5 m/s and 2 m/s respectively. For the number of people (users), we chose 100, 200 and 300 where we created 40 scenarios for each for statistical significance.

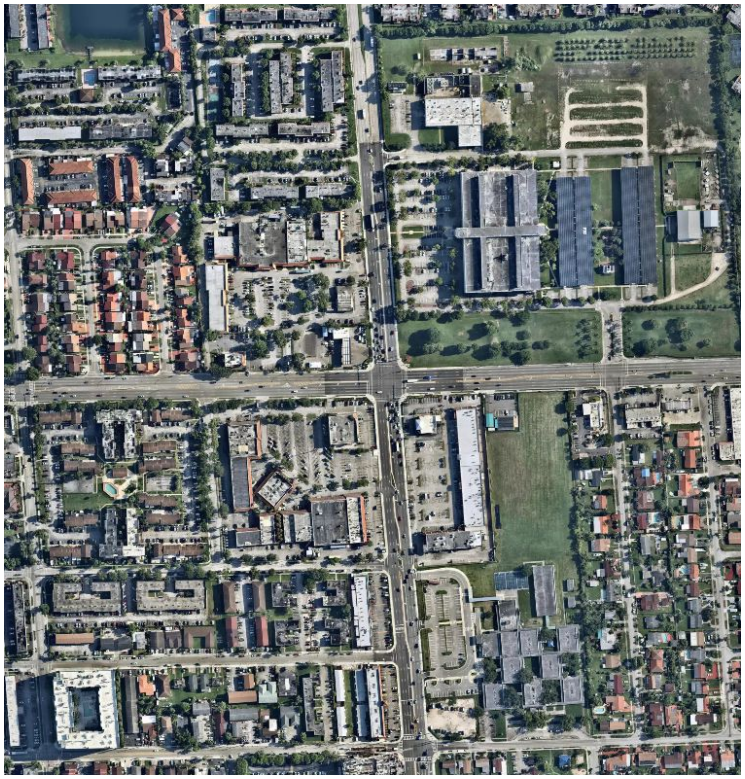

Figure 6.6: The map of FIU's Engineering Center and its surroundings

Next, we preprocessed BonnMotion scenario files. The scenarios created by BonnMotion have different time values for each node as can be seen in Table 6.2. In order to get the network topology at a specific time, we need all the nodes to have the identical time values. To achieve this, we grabbed the x-y coordinates of

each node every 10 minutes from the scenario file and recorded the resulting data in a new *.csv* file. In this way, every node had *6 hours/10 minutes = 36* rows in the preprocessed scenario file.

With the preprocessed scenarios at hand, we could then create the mesh and LN topologies for the simulation. Each preprocessed scenario has 36 mesh topologies in it. To calculate a mesh topology from a scenario file for a given time, we first need to calculate the distances between all the nodes for all times. This is a costly operation to do at simulation run time, thus we opted for precomputing these distances and saving them into a *.csv* file for each scenario. In this way, we can just read these files in the simulation run time and calculate the mesh topologies much faster. After this step, we also created the mobility-aware mesh topology for each scenario using Algorithm 3. For the value of d, we used 90 meters to keep a conservative value for a potential IEEE 802.11n/ac coverage considering urban environments [EA22]. For k, we tried the following values: (*2, 3, 4, 5, 6, 7, 8, 9, 10*). When k was greater than 5, we started getting disconnected graphs for mobility-aware mesh topology. Values less than 5 on the other hand generated topologies with too many links which is not ideal. Thus, we chose 5 for the value of k. However, still we got 3 scenarios that did not include some nodes in the mobility-aware mesh topology thus, we performed the experiments with 37 scenarios instead.

6.3.2 Simulation Setup, Metrics and Baselines

Simulation Setup: Our aim in the simulations is to perform N arbitrary LN payments every 10 minutes between M nodes. Amount A of a payment is arbitrarily selected from the following list of values: (*1, 5, 10, 20, 50, 100*) satoshi. Across the simulations, we varied N with the following list of values: (*20, 30, 40, 50,*

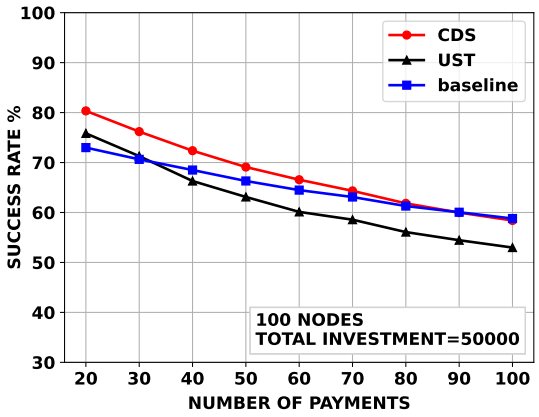

Figure 6.7: Success rate for 100 nodes.

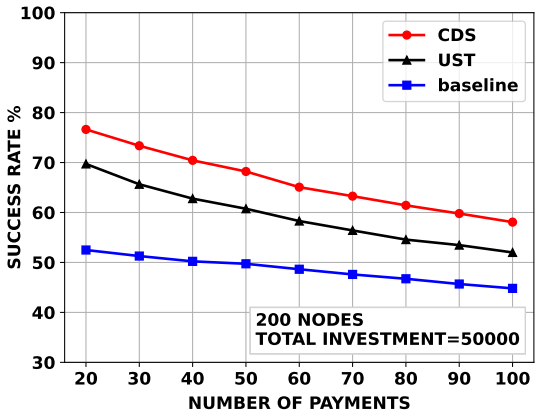

Figure 6.8: Success rate for 200 nodes.

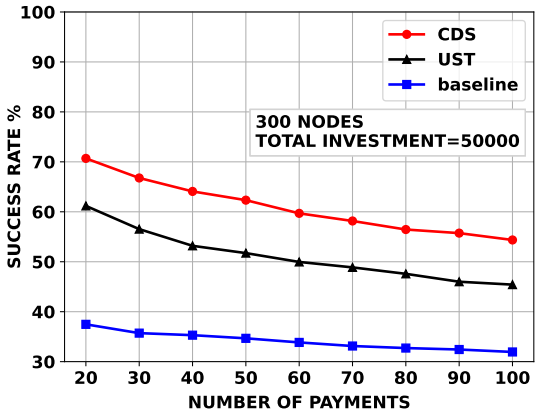

Figure 6.9: Success rate for 300 nodes.

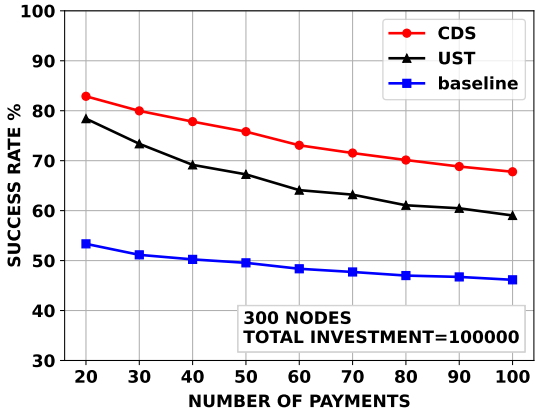

Figure 6.10: Success rate for 100000 satoshi.

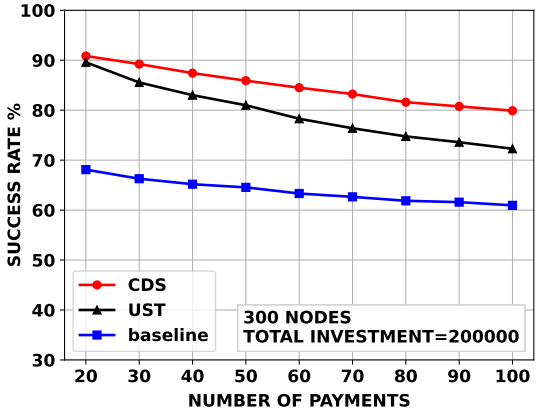

Figure 6.11: Success rate for 200000 satoshi.

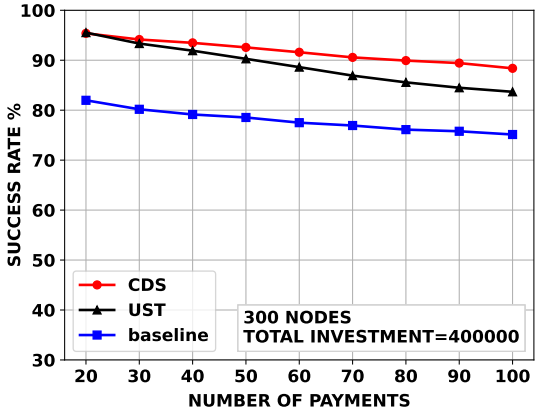

Figure 6.12: Success rate for 400000 satoshi.

60, 70, 80, 90, 100). For the number of nodes M in the simulations, we tested the following values: (*100, 200, 300*). Basically, they match the number of nodes we initially input to the MSLAW algorithm during the scenario generation. As mentioned before, we used 37 scenarios for each value of M. To have a fair comparison across the simulations, the random selection of N payments with random amounts A was done using fixed seed values. For channel capacities, we equally distributed a *total investment amount* to all the channels in a simulation. The following total investment amounts were used in the simulations: (*50000, 100000, 200000, 300000, 400000*) satoshi. For example, if there were 100 channels in a specific simulation and the total investment amount was 100,000 satoshi, then each channel was funded with 1,000 satoshi. Essentially, depending on the total investment and the total number of channels, channel capacities across the simulations vary. The reason for keeping the total investment amount fixed across simulations is to see the effect of other simulation parameters more clearly and fairly. For the MCDS algorithm, we used an existing Python implementation[3] of the non-distributed MCDS algorithm proposed by Rai et al. [RGVT09]. For generating USTs, we used *DPPy* package in Python[4] which has an implementation of the Wilson's algorithm [PW98] for general graphs.

Metrics: We use the *payment success rate* as our main metric on LN topologies which shows the percentage of transactions that were successfully received by the recipients. This metric depends on many parameters such as the total investment amount, number of payments every 10 minutes, number of nodes in the network and more importantly the selected channel assignment approach.

[3] `http://sparkandshine.net/en/calculate-connected-dominating-sets-cds/`

[4] `https://dppy.readthedocs.io/en/latest/exotic_dpps/ust.html`

Baseline: To test the effectiveness of the CDS and ST approaches, we need a baseline approach. The most intuitive candidate was the mobility-aware mesh topology which can be directly used as an LN topology. Thus, for the baseline approach, we opened an LN channel for each link in the mobility-aware mesh topology of the respective mobility scenario.

6.3.3 Simulation Results

We conducted several experiments whose results are shown in Fig. 6.7-6.12. We discuss each scenario in details below:

Impact of the number of nodes: Looking at Figures 6.7, 6.8, 6.9, we can see that increasing the number of nodes in the network while keeping total investment fixed negatively affects the success rate. This is because, when there are more nodes in the network, there are also more channels. Since the total investment is kept the same, the channel capacities are lower on the network with more nodes. Thus, if we do not want to sacrifice on the success rate, we need to invest more money into the network when there are more users. This is actually not an issue in real life since each new user fund their channels with their own funds. Essentially, total investment automatically increases when a new user joins the network. Regardless of the number of nodes, increased number of payments deplete the channels quickly and thus there is a reduction in success rate for all approaches.

Impact of total investment: We can see from Figures 6.10, 6.11, 6.12 that increasing the total investment increases the success rates for all approaches dramatically. Specifically, when the total investment was increased from 50000 satoshi to 400000 for 300 nodes, the success rates reached up to 95% for CDS, 95% for UST, and 82% for the baseline approach. Such increase in the success rates makes a lot

of sense because by increasing the total investment, we are essentially putting more funds in each channel which helps the channels last longer and do not get depleted as quick. We see that even when the total investment is low (i.e., 50,000 satoshi), more than half of the payments in the network are successful for almost all cases.

Impact of using CDS and UST: The most interesting outcome of the results are on the used approach. We can see from almost all the results that CDS consistently performs the best while UST comes after CDS and baseline approach performs the worst. To understand why CDS and UST have much higher success rate than the baseline, we need to look at the number of channels in each case which are shown in Table 6.3. These results are an average from 37 different mobility scenarios we used for the simulations.

As can be seen, baseline approach has too many channels compared to the CDS and UST. Since, we are keeping the total investment constant, the channel capacities in baseline experiments are very low compared to CDS and UST experiments. This in turn causes the payments to fail frequently due to insufficient capacity on the channels. The only exception was the case with Fig. 6.7 where UST performed worse than baseline even though its number of channels is lower. This can be attributed to the fact that there are much less users and alternative routes in this particular case. The channel capacities can be quickly depleted using same the routes for CDS and UST especially when the number of transactions increase. In case of baseline, there may be other routes that still allow successful transactions. When we increase the number of users to 300 in Fig. 6.10, we see that the trends change completely since there are more users to offer more routes and thus baseline approach suffers from low channel capacities. Thus, it can be concluded that having fewer number of channels with higher capacities is more effective.

Table 6.3: Average Number of Channels for Different Approaches from 37 Different Scenarios

	Average Number of Channels		
Approach	**100 nodes**	**200 nodes**	**300 nodes**
CDS	99	199	299
UST	99	199	299
Baseline	665	2876	6381

In general, when we compare CDS and UST, we see that CDS has higher success rate than UST in all experiments. To understand why this is the case, we checked several properties of the CDS and UST graphs. First thing we noticed was CDS graphs have vertices with high degrees (i.e., vertices with many edges) while the vertex degrees in UST graphs were almost uniform. This causes payments to take longer paths in UST case which results in payments to fail more frequently due to channels with insufficient capacities along the payment path. CDS case on the other hand essentially has vertices acting as hubs which can relay the payments to recipients. As we predicted, this can be attributed to the *closeness centrality* (CC) feature for CDS-based topologies which measures how short the shortest paths are from a node i to all other nodes: $CC(i) = N - 1/\sum_j d(i,j)$ where $d(i,j)$ is the length of the shortest path between nodes i and j and N is the number of nodes. Due to the way we generate CDS-based topologies, there are hub nodes which enable closeness centrality to be higher.

Analysis of failed payments: To better understand the performance difference between CDS, UST and baseline approaches, we also analyzed the percentage for failed payments in Table 6.4. These results presented are cumulatively calculated from all of the simulations. Total number of payments for each case was 3,596,400.

Table 6.4: Percentages of Payment Failures for a Total of 3,596,400 Payments from 45 Simulations

	Not Enough Capacity			**No Mesh Path**		
Approach	**100 nodes**	**200 nodes**	**300 nodes**	**100 nodes**	**200 nodes**	**300 nodes**
CDS	15.45%	16.27%	19.2%	2.71%	2.43%	2.18%
UST	19.14%	20.83%	25.38%	2.86%	2.63%	2.47%
Baseline	15.17%	26.77%	40.20%	2.26%	1.75%	1.35%

Looking at the results, first thing we notice is that most payments fail because of not having enough capacity on the channels. The rest of the failures are due to not having paths on the mesh topology itself which are insignificant (i.e., only comprise 2-3% of all payments and approaches to zero when we have more nodes). Note that we do not have failures due to not having paths on the LN topologies since we made sure all LN topologies we generated are connected.

Unarguably, the most important observation from these results is the effectiveness of CDS and UST approaches on reducing the payment failures from not having enough channel capacities. We see an improvement around up to 50% over the baseline. Overall, these results show that with the right number of nodes and investment amount, we can achieve a success rate close to 100% when using CDS-based approach.

CHAPTER 7

CONCLUSION AND FUTURE WORK

LN has been formed as a new payment network to address the drawbacks of Bitcoin transactions in terms of time and cost. In addition to the fast and cheap transactions, LN provides a perfect opportunity for covert communications as no transactions are recorded in the blockchain. Therefore, in this dissertation, we first proposed a new covert hybrid botnet by utilizing the LN payment network formed for Bitcoin operations. The idea was to control the C&C servers through commands that are sent in the form of LN payments. The proof-of-concept implementation of this architecture indicated that LNBot can be successfully created and commands for attacks can be sent to C&C servers through LN with negligible costs yet with very high anonymity. To improve on the cost and delay of sending the commands, we proposed a slightly modified version of LNBot which utilizes a new feature of LN that enables sending messages by embedding them into payments. Our evaluation results showed an 98% improvement in both cost and delay metrics. Finally, we proposed D-LNBot, a cost-free and distributed version of LNBot with faster command sending times. Our evaluation results showed that, commands are propagated to all the C&C servers in $O(mlogn)$ time compared to LNBot's $O(na)$. To minimize the impact of these proposed botnets, we offered several countermeasures that include the possibility of searching for the botmaster.

In our second work, we proposed a secure and efficient protocol for enabling IoT devices to use Bitcoin's LN for sending and receiving payments. By introducing (2,2)-threshold scheme to LN and modifying LN's existing peer protocol for channel management and commitment transactions, a third peer (i.e., IoT device) was added to the LN channels. The purpose was to enable resource-constrained IoT devices that normally cannot interact with LN to interact with it and perform

micro-payments with other users. IoT device's interactions with LN are achieved through an untrusted gateway node that has access to LN and thus can provide LN services in return for a fee. A game theoretic security analysis was provided to prove that the IoT device does not lose money when other channel peers attempt to cheat. Our evaluation results showed that the proposed protocol is scalable and enables IoT devices to use LN with negligible delay, cost, networking and energy consumption overheads.

Finally, in the last work, we proposed using mobile community wireless mesh networks to power offline LN payments so that people in a community/neighborhood without active Internet connections can continue transacting among themselves. We showed the feasibility of such a setup on a IP-over-BLE star network and a full WiFi mesh network using 8 Raspberry Pis. Our experiments implied that the way the channels are opened have a huge impact on the overall success rate of the payments in the network. Additionally, assuming that the mesh users can also move around complicates channel opening even more. Thus, we proposed two channel assignment approaches taking into account the mobility of the users. First approach is based on connected dominating set concept and the second one utilizes uniform spanning tree. To test these approaches in a large-scale, we implemented a simulator and extensively analyzed overall success rate of the payments on different settings. Our results showed that, our proposed approaches work well and can achieve success rates up to 95% on large mobile wireless mesh networks.

We list the potential future research to extend these works below:

- In D-LNBot, we examined the total command propagation time using a Python simulator. A more realistic way to realize it would be by using Bitcoin's Simnet. In Simnet, we can mine the blocks ourselves and deploy the nodes in a programmatic way.

- In LNGate^2, thresholdizing the commitment points were left as a future work. It prevents nodes from single-handedly revealing their revocation keys before the channel state is updated. The implementation requires a new cryptographic protocol since channel states utilize hash chains.
- In LNGate^2, the security analysis section would benefit from a game theoretic security analysis of the ransom attacks similar to the analysis of collusion attacks. This is mainly to make the security analysis more formal.
- In LNMesh, we assumed that all nodes go offline and come back online at the same time which makes sure that nodes cannot cheat against each other. If this assumption is relaxed (i.e., some nodes go online before others), then there will be a need to analyze and discuss the security of the scheme covering more possibilities.
- In LNMesh, alternative channel assignment approaches other than spanning trees can be tested to see if overall success rates can be increased even more. Additionally, the effect of other parameters such as wireless coverage can be investigated.
- There may be many other use cases of LN that need to be explored for their feasibility and performance. Therefore, there is a need to identify potential use cases of LN. This may include using it for any covert communication purposes, community tools, web3 applications, and so on.

BIBLIOGRAPHY

[ABC+18] Nicola Atzei, Massimo Bartoletti, Tiziana Cimoli, Stefano Lande, and Roberto Zunino. SoK: Unraveling Bitcoin smart contracts. In *Principles of Security and Trust*, pages 217–242. Springer, 2018.

[ACI23] ACINQ. Phoenix is a self-custodial Bitcoin wallet using lightning to send/receive payments., 2023. `https://github.com/ACINQ/phoenix`.

[AEGPS10] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla, and Matthias Schwamborn. BonnMotion: a mobility scenario generation and analysis tool. In *3rd International ICST Conference on Simulation Tools and Techniques*. ICST, 2010.

[AM22] Omar Alibrahim and Majid Malaika. Botract: abusing smart contracts and blockchain for botnet command and control. *International Journal of Information and Computer Security*, 17(1-2):147–163, 2022.

[AMLH15] Syed Taha Ali, Patrick McCorry, Peter Hyun-Jeen Lee, and Feng Hao. ZombieCoin: Powering next-generation botnets with Bitcoin. In *Financial Cryptography and Data Security*, pages 34–48. Springer, 2015.

[AMLH18] Syed Taha Ali, Patrick McCorry, Peter Hyun-Jeen Lee, and Feng Hao. ZombieCoin 2.0: managing next-generation botnets using Bitcoin. *International Journal of Information Security*, 17(4):411–422, 2018.

[App73] G. M. Appa. The transportation problem and its variants. *Journal of the Operational Research Society*, 24(1):79–99, 1973.

[Aum95] Robert J. Aumann. Backward induction and common knowledge of rationality. *Games and Economic Behavior*, 8(1):6–19, 1995.

[BAA+22] Shrenik Bhansali, Ahmet Aris, Abbas Acar, Harun Oz, and A. Selcuk Uluagac. A first look at code obfuscation for WebAssembly. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 140–145. Association for Computing Machinery, 2022.

[BAS+19] Leon Böck, Nikolaos Alexopoulos, Emine Saracoglu, Max Mühlhäuser, and Emmanouil Vasilomanolakis. Assessing the threat of blockchain-based botnets. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–11. IEEE, 2019.

[BCJ+09] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A survey of botnet technology and defenses. In *2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 299–304. IEEE, 2009.

[BFG+03] Hichem Baala, Olivier Flauzac, Jaafar Gaber, Marc Bui, and Tarek El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *Journal of Parallel and Distributed Computing*, 63(1):97–104, 2003.

[BFTFPS19] Mathis Baden, Christof Ferreira Torres, Beltran Borja Fiz Pontiveros, and Radu State. Whispering botnet command and control instructions. In *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 77–81. IEEE, 2019.

[Bit19] BitMEX Research. The lightning network (part 2) – routing fee economics, 2019. `https://blog.bitmex.com/the-lightning-network-part-2-routing-fee-economics/`.

[Blo23a] Blockstream. Core lightning (CLN): A specification compliant lightning network implementation in C, 2023. `https://github.com/ElementsProject/lightning`.

[Blo23b] Blockstream. PeerSwap, 2023. `https://github.com/ElementsProject/peerswap`.

[Bra18] Brainbot Labs. μraiden - a payment channel framework for fast & free off-chain ERC20 token transfers, 2018. `https://raiden.network/micro.html`.

[Bro10] Dennis Brown. Resilient botnet command and control with Tor, 2010. `https://defcon.org/images/defcon-18/dc-18-presentations/D.Brown/DEFCON-18-Brown-TorCnC.pdf`.

[Bro21] Ryan Browne. Bitcoin's wild rally — and a fear of missing out — has retail investors flocking to crypto, 2021. `https://www.cnbc.com/2`

021/01/08/bitcoin-rally-fomo-has-retail-investors-flocking-to-crypto.html.

[BRV+22] Sriramulu Bojjagani, PV Venkateswara Rao, Dinesh Reddy Vemula, B Ramachandra Reddy, and T Jaya Lakshmi. A secure IoT-based micro-payment protocol for wearable devices. *Peer-to-Peer Networking and Applications*, 15(2):1163–1188, 2022.

[CJCLB12] Telvis E. Calhoun Jr, Xiaojun Cao, Yingshu Li, and Raheem Beyah. An 802.11 MAC layer covert channel. *Wireless Communications and Mobile Computing*, 12(5):393–405, 2012.

[CM14] Matteo Casenove and Armando Miraglia. Botnet over Tor: The illusion of hiding. In *2014 6th International Conference On Cyber Conflict (CyCon 2014)*, pages 273–282. IEEE, 2014.

[CXS+18] Wubing Chen, Zhiying Xu, Shuyu Shi, Yang Zhao, and Jun Zhao. A survey of blockchain applications in different domains. In *Proceedings of the 2018 International Conference on Blockchain Technology and Application*, page 17–21. ACM, 2018.

[Dec22] Christian Decker. Data about the past and current structure of the lightning network, 2022. https://github.com/lnresearch/topology.

[Dec23] Christian Decker. The noise plugin, 2023. https://github.com/lightningd/plugins/tree/master/noise.

[Des94] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[DG09] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *2009 30th IEEE Symposium on Security and Privacy*, pages 269–282. IEEE, 2009.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DNY17] Alexandra Dmitrienko, David Noack, and Moti Yung. Secure wallet-assisted offline Bitcoin payments with double-spender revocation. In

Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pages 520–531. ACM, 2017.

[EA22] Iman ElKassabi and Atef Abdrabou. An experimental comparative performance study of different WiFi standards for smart cities outdoor environments. In *2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0450–0455. IEEE, 2022.

[FAD20] Federico Franzoni, Ivan Abellan, and Vanesa Daza. Leveraging Bitcoin testnet for bidirectional botnet command and control systems. In *Financial Cryptography and Data Security*, pages 3–19. Springer, 2020.

[FAZ18] Davor Frkat, Robert Annessi, and Tanja Zseby. ChainChannels: Private botnet communication over public blockchains. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1244–1252. IEEE, 2018.

[FCFL18] Tiago M Fernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the Internet of things. *IEEE Access*, 6:32979–33001, 2018.

[Fre23] Freifunk. Better approach to mobile ad-hoc networking (B.A.T.M.A.N.), 2023. `https://www.open-mesh.org/projects/open-mesh/wiki`.

[GK98] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[Gol13] Jennifer Golbeck. Chapter 3 - network structure and measures. In *Analyzing the Social Web*, pages 25–44. Morgan Kaufmann, 2013.

[GRSB18] Sharon Goldberg, Leonid Reyzin, Omar Sagga, and Foteini Baldimtsi. Certifying RSA public keys with an efficient NIZK. Cryptology ePrint Archive, Report 2018/057, 2018.

[GSN+07] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-Peer bot-

nets: Overview and case study. In *First Workshop on Hot Topics in Understanding Botnets (HotBots 07)*. USENIX Association, 2007.

[HJ19] Christopher Hannon and Dong Jin. Bitcoin payment-channels for resource limited IoT devices. In *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, pages 50–57. ACM, 2019.

[HT11] Jonathan Hui and Pascal Thubert. Compression format for IPv6 datagrams over IEEE 802.15.4-based networks. Technical report, 2011. `https://www.rfc-editor.org/rfc/rfc6282.html`.

[Huf06] David A. Huffman. A method for the construction of minimum-redundancy codes. *Resonance*, 11(2):91–99, 2006.

[Jag19] Joost Jager. lncli+invoices: experimental key send mode, 2019. `https://github.com/lightningnetwork/lnd/pull/3795`.

[KAC12] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 906–917. ACM, 2012.

[KEC+20] Ahmet Kurt, Enes Erdin, Mumin Cebe, Kemal Akkaya, and A Selcuk Uluagac. LNBot: A covert hybrid botnet on Bitcoin lightning network for fun and profit. In *Computer Security – ESORICS 2020*, pages 734–755. Springer, 2020.

[KMS+21] Ahmet Kurt, Suat Mercan, Omer Shlomovits, Enes Erdin, and Kemal Akkaya. LNGate: Powering IoT with next generation lightning micropayments using threshold cryptography. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 117–128. ACM, 2021.

[KSAU21] Ahmet Kurt, Nico Saputro, Kemal Akkaya, and A Selcuk Uluagac. Distributed connectivity maintenance in swarm of drones during post-disaster transportation applications. *IEEE Transactions on Intelligent Transportation Systems*, 22(9):6061–6073, 2021.

[KSWK21] Dimitri Kamenski, Arash Shaghaghi, Matthew Warren, and Salil S. Kanhere. Attacking with Bitcoin: Using Bitcoin to build resilient botnet armies. In *13th International Conference on Computational*

Intelligence in Security for Information Systems (CISIS 2020), pages 3–12. Springer, 2021.

[Kur21] Kenichi Kurimoto. Lightning network x IoT(LoT); potential, challenges and solutions, 2021. https://medium.com/nayuta-en/lightning-network-x-iot-lot-potential-challenges-and-solutions-6e4d8b4c252a.

[Kuw19] Ichiro Kuwahara. Operating lightning #0001 — basic challenges in operating lighting network nodes, 2019. https://medium.com/crypto-garage/operating-lightning-0001-basic-challenges-in-operating-lighting-network-nodes-d04386ac931a.

[LFX+20] Dunfeng Li, Yong Feng, Yao Xiao, Mingjing Tang, and Xiaodong Fu. A data trading scheme based on payment channel network for Internet of things. In *Blockchain and Trustworthy Systems*, pages 319–332. Springer, 2020.

[Lig22a] Lightning Labs. BOLT #2: Peer protocol for channel management, 2022. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md.

[Lig22b] Lightning Labs. Onion routed micropayments for the lightning network, 2022. https://github.com/lightningnetwork/lightning-onion.

[Lig23a] Lightning Labs. Lightning network daemon, 2023. https://github.com/lightningnetwork/lnd.

[Lig23b] Lightning Labs. Neutrino: Privacy-preserving Bitcoin light client, 2023. https://github.com/lightninglabs/neutrino.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology – CRYPTO 2017*, pages 613–644. Springer, 2017.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1837–1854. ACM, 2018.

[LWZ+21] Rujia Li, Qin Wang, Xinrui Zhang, Qi Wang, David Galindo, and Yang Xiang. An offline delegatable cryptocurrency system. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2021.

[MR88] John Moore and Rafael Repullo. Subgame perfect implementation. *Econometrica*, 56(5):1191–1220, 1988.

[Mye78] Roger B. Myerson. Refinements of the Nash equilibrium concept. *International Journal of Game Theory*, 7(2):73–80, 1978.

[Mye19] Richard Myers. Lot49: A lightweight protocol to incentivize mobile peer-to-peer communication. Technical report, 2019.

[MZ21] Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. In *Financial Cryptography and Data Security*, pages 170–188. Springer, 2021.

[NHP+11] Shishir Nagaraja, Amir Houmansadr, Pratch Piyawongwisal, Vijit Singh, Pragya Agarwal, and Nikita Borisov. Stegobot: A covert social network botnet. In *Information Hiding*, pages 299–313. Springer, 2011.

[NWG+20] Kervins Nicolas, Yi Wang, George C Giakos, Bingyang Wei, and Hongda Shen. Blockchain system defensive overview for double-spend and selfish mining attacks: A systematic approach. *IEEE Access*, 9:3838–3857, 2020.

[OALU22] Harun Oz, Ahmet Aris, Albert Levi, and A. Selcuk Uluagac. A survey on ransomware: Evolution, taxonomy, and defense solutions. *ACM Computing Surveys*, 54(11s), 2022.

[PAL20] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. TinyEVM: Off-chain smart contracts on low-power IoT devices. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 507–518. IEEE, 2020.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin lightning network: Scalable off-chain instant payments. Technical report, 2016. `http://lightning.network/lightning-network-paper.pdf`.

[PH15] Nick Pantic and Mohammad I. Husain. Covert botnet command and control using Twitter. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 171–180. ACM, 2015.

[PP18] Antonio Pirozzi and Pierluigi Paganini. Experts presented BOTCHAIN, the first fully functional botnet built upon the Bitcoin protocol, 2018. `https://securityaffairs.co/wordpress/77395/malware/botchain-botnet-bitcoin-protocol.html`.

[PW98] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, 27(2):170–217, 1998.

[PW19] Arman Pouraghily and Tilman Wolf. A lightweight payment verification protocol for blockchain transactions on IoT devices. In *2019 International Conference on Computing, Networking and Communications (ICNC)*, pages 617–623. IEEE, 2019.

[RDS22] Abhimanyu Rawat, Vanesa Daza, and Matteo Signorini. Offline scaling of IoT devices in IOTA blockchain. *Sensors*, 22(4), 2022.

[RG14] Douglas Roffel and Christopher Garrett. A novel approach for computer worm control using decentralized data structures. University of California Santa Cruz Class Project, 2014.

[RGVT09] M. Rai, N. Garg, S. Verma, and S. Tapaswi. A new heuristic approach for minimum connected dominating set in adhoc wireless networks. In *2009 IEEE International Advance Computing Conference*, pages 284–289. IEEE, 2009.

[RKG20] Jérémy Robert, Sylvain Kubler, and Sankalp Ghatpande. Enhanced lightning network (off-chain)-based micropayment in IoT ecosystems. *Future Generation Computer Systems*, 112:283–296, 2020.

[RPBdAD22] Gabriel Antonio F Rebello, Maria Potop-Butucaru, Marcelo Dias de Amorim, and Otto Carlos MB Duarte. Securing wireless payment-channel networks with minimum lock time windows. In *ICC 2022-IEEE International Conference on Communications*, pages 2297–2302. IEEE, 2022.

[RSD+19] Ken Reese, Trevor Smith, Jonathan Dutson, Jonathan Armknecht, Jacob Cameron, and Kent Seamons. A usability study of five two-factor authentication methods. In *Proceedings of the Fifteenth USENIX Conference on Usable Privacy and Security*, page 357–370. USENIX Association, 2019.

[SA13] Matthias Schwamborn and Nils Aschenbruck. Introducing geographic restrictions to the SLAW human mobility model. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 264–272. IEEE, 2013.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SMG19] Sarwar Sayeed and Hector Marco-Gisbert. Assessing blockchain consensus and security mechanisms against the 51% attack. *Applied Sciences*, 9(9), 2019.

[SMP18] Saleh Soltan, Prateek Mittal, and H. Vincent Poor. BlackIoT: IoT botnet of high wattage devices can disrupt the power grid. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 15–32. USENIX Association, 2018.

[SN15] Amirali Sanatinia and Guevara Noubir. OnionBots: Subverting privacy infrastructure for cyber attacks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 69–80. IEEE, 2015.

[SS23] Cosimo Sguanci and Anastasios Sidiropoulos. Mass exit attacks on the lightning network. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–3. IEEE, 2023.

[SSN+19] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles A. Kamhoua, DaeHun Nyang, and Aziz Mohaisen. *Overview of Attack Surfaces in Blockchain*, chapter 3, pages 51–66. Wiley, 2019.

[SSPS13] Sérgio S.C. Silva, Rodrigo M.P. Silva, Raquel C.G. Pinto, and Ronaldo M. Salles. Botnets: A survey. *Computer Networks*, 57(2):378–403, 2013.

[SSSA11] Mustafa Y. Sir, Izzet F. Senturk, Esra Sisikoglu, and Kemal Akkaya. An optimization-based approach for connecting partitioned mobile sensor/actuator networks. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 525–530. IEEE, 2011.

[Sub18] William Suberg. First Bitcoin mainnet lightning network product launches as developers net $2.5 mln, 2018. `https://cointelegraph.com/news/first-bitcoin-mainnet-lightning-network-product-launches-as-developers-net-25-mln`.

[Swe17] Jonny Sweeny. Botnet resiliency via private blockchains, 2017. `https://www.sans.org/white-papers/38050/`.

[TAK+15] Zisis Tsiatsikas, Marios Anagnostopoulos, Georgios Kambourakis, Sozon Lambrou, and Dimitris Geneiatakis. Hidden in plain sight. SDP-based covert channel for botnet communication. In *Trust, Privacy and Security in Digital Business*, pages 48–59. Springer, 2015.

[Tea18] Team KZen. Bitcoin wallet powered by two-party ECDSA - extended abstract. Technical report, 2018. `https://github.com/ZenGo-X/gotham-city/blob/master/white-paper/white-paper.pdf`.

[TGL+20] Jing Tian, Gaopeng Gou, Chang Liu, Yige Chen, Gang Xiong, and Zhen Li. DLchain: A covert channel over blockchain based on dynamic labels. In *Information and Communications Security*, pages 814–830. Springer, 2020.

[The23] The Bitcoin Core Project. Bitcoin core integration/staging tree, 2023. `https://github.com/bitcoin/bitcoin`.

[TO20] Taisei Takahashi and Akira Otsuka. Short paper: secure offline payments in Bitcoin. In *Financial Cryptography and Data Security*, pages 12–20. Springer, 2020.

[TYL+20] Nachiket Tapas, Yechiav Yitzchak, Francesco Longo, Antonio Puliafito, and Asaf Shabtai. P4UIoT: Pay-per-piece patch update delivery for IoT using gradual release. *Sensors*, 20(7), 2020.

[vWOvD18] Rolf van Wegberg, Jan-Jaap Oerlemans, and Oskar van Deventer. Bitcoin money laundering: mixed results? an explorative study on

money laundering of cybercrime proceeds using Bitcoin. *Journal of Financial Crime*, 25(2):419–435, 2018.

[VZF17] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. Botnet communication patterns. *IEEE Communications Surveys & Tutorials*, 19(4):2768–2796, 2017.

[WC04] Bang Ye Wu and Kun-Mao Chao. *Spanning trees and optimization problems*. Chapman and Hall/CRC, 2004.

[WCZ20] Hua Wang, Jinli Cao, and Yanchun Zhang. *Untraceable Electronic Cash System in the Internet of Things*, pages 43–63. Springer, 2020.

[WLC+22] Zhi Wang, Chaoge Liu, Xiang Cui, Jie Yin, Jiaxi Liu, Di Wu, and Qixu Liu. DeepC2: AI-powered covert command and control on OSNs. In *Information and Communications Security*, pages 394–414. Springer, 2022.

[WLH+21] Min-Hao Wu, Chia-Hao Lee, Fu-Hau Hsu, Kai-Wei Chang, Tsung-Huang Huang, Ting-Cheng Chang, and Li-Min Yi. Simple and ingenious mobile botnet covert network based on adjustable unit (SIMBAIDU). *Mathematical Problems in Engineering*, 2021, 2021.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, 2014. `https://github.com/ethereum/yellowpaper`.

[WSZ10] Ping Wang, Sherri Sparks, and Cliff C. Zou. An advanced hybrid Peer-to-Peer botnet. *IEEE Transactions on Dependable and Secure Computing*, 7(2):113–127, 2010.

[WZWX21] Qingtao Wang, Chi Zhang, Lingbo Wei, and Yankai Xie. HyperChannel: A secure layer-2 payment network for large-scale IoT ecosystem. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[YCL+20] Jie Yin, Xiang Cui, Chaoge Liu, Qixu Liu, Tao Cui, and Zhi Wang. CoinBot: A covert botnet in the cryptocurrency network. In *Information and Communications Security*, pages 107–125. Springer, 2020.

[Zen23] ZenGo X. Minimalist Bitcoin decentralized HD wallet using 2 party ECDSA, 2023. `https://github.com/ZenGo-X/gotham-city`.

[ZHZB20] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *IEEE Access*, 8:16440–16455, 2020.

[Zoh18] Omer Zohar. Unblockable Chains, 2018. `https://sector.ca/sessions/unblockable-chains-is-blockchain-the-ultimate-malicious-infrastructure/`.

[ZZZ+19] Yi Zhong, Anmin Zhou, Lei Zhang, Fan Jing, and Zheng Zuo. DUST-Bot: A duplex and stealthy P2P-based botnet in the Bitcoin network. *PLOS ONE*, 14(12):1–27, 2019.

VITA

AHMET KURT

2018 B.S., Electrical and Electronics Engineering
Antalya Bilim University
Antalya, Turkey

2018 B.S., Computer Engineering
Antalya Bilim University
Antalya, Turkey

2021 M.Sc., Electrical Engineering
Florida International University
Miami, Florida

2023 Ph.D., Electrical and Computer Engineering
Florida International University
Miami, Florida

PUBLICATIONS AND PRESENTATIONS

• A. Kurt, E. Erdin, K. Akkaya, A. S. Uluagac, M. Cebe, (2023). *D-LNBot: A Scalable, Cost-Free and Covert Hybrid Botnet on Bitcoin's Lightning Network*, in IEEE Transactions on Dependable and Secure Computing, 1-18.

• A. Kurt, K. Akkaya, S. Yilmaz, S. Mercan, O. Shlomovits, E. Erdin, (2023). *LNGate^2: Secure Bidirectional IoT Micro-payments using Bitcoin's Lightning Network and Threshold Cryptography*, in arXiv preprint arXiv:2206.02248, 1-17.

• A. Kurt, A. Sahin, R. Harrilal-Parchment, K. Akkaya, (2023). *LNMesh: Who Said You need Internet to send Bitcoin? Offline Lightning Network Payments using Community Wireless Mesh Networks*, in Proceedings of 2023 IEEE 24rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 1-10.

• A. Bhattarai, M. Veksler, A. Sahin, A. Kurt, K. Akkaya, (2023). *Crypto Wallet Artifact Detection on Android Devices using Advanced Machine Learning Techniques*, in Proceedings of International Conference on Digital Forensics and Cyber Crime (ICDF2C 2022), 111-132.

• S. Mercan, A. Kurt, E. Erdin, K. Akkaya, (2022). *Cryptocurrency Solutions to Enable Micro-payments in Consumer IoT*, in IEEE Consumer Electronics Magazine, 11(2):97-103.

• A. Kurt, S. Mercan, E. Erdin, K. Akkaya, (2021). *3-of-3 Multisignature Approach for Enabling Lightning Network Micro-payments on IoT Devices*, in ITU Journal on Future and Evolving Technologies, 2(5):53-67.

• A. Kurt, S. Mercan, O. Shlomovits, E. Erdin, K. Akkaya, (2021). *LNGate: Powering IoT with Next Generation Lightning Micro-payments using Threshold Cryptography*, in Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), 117-128.

• A. Kurt, S. Mercan, E. Erdin, K. Akkaya, (2021). *Enabling Micro-payments on IoT Devices using Bitcoin Lightning Network*, in Proceedings of 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 1-3.

• A. Kurt, N. Saputro, K. Akkaya, A. S. Uluagac, (2021). *Distributed Connectivity Maintenance in Swarm of Drones During Post-Disaster Transportation Applications*, in IEEE Transactions on Intelligent Transportation Systems, 22(9):6061-6073.

• A. Kurt, E. Erdin, M. Cebe, K. Akkaya, A. S. Uluagac, (2020). *LNBot: A Covert Hybrid Botnet on Bitcoin Lightning Network for Fun and Profit*, in Proceedings of 25th European Symposium on Research in Computer Security (ESORICS 2020), 734-755.

• A. Kurt, K. Akkaya, (2020). *Connectivity Maintenance Extensions to IEEE 802.11s MAC Layer in ns-3*, in Proceedings of the 2020 Workshop on ns-3 (WNS3 '20), 17-24.