Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Erik Turøy Midtun

Bootstrapping decentralized overlay networks

Master's thesis in Communication Technology and Digital Security
Supervisor: Lasse Øverlier
June 2023

NTNU
Norwegian University of Science and Technology

Erik Turøy Midtun

Bootstrapping decentralized overlay networks

Master's thesis in Communication Technology and Digital Security
Supervisor: Lasse Øverlier
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title:	Bootstrapping decentralized overlay networks
Student:	Midtun, Erik Turøy

Problem description:

Decentralized overlay networks offer a robust, scalable, and fault-tolerant way of creating distributed applications. They can be constructed on top of existing networks, thereby inheriting the underlying network's properties. This was done in a previous master's thesis where an overlay network was built on the anonymous Tor network, thus providing the overlay network with anonymity.

Bootstrapping is the mechanism that involves discovering, initiating, and establishing connections between nodes in a network. Creating such mechanisms can be a significant challenge, particularly depending on the network's required properties.

This thesis aims to provide an overview of the challenges associated with bootstrapping networks, examine how these issues were resolved in other networks, and then create an improved bootstrapping mechanism for the decentralized overlay network built on top of Tor.

Approved on:	2023-02-16
Supervisor:	Øverlier, Lasse, NTNU

Abstract

This master's thesis is a continuation of the groundwork established in a prior thesis by Fallang [1], where most of the fundamental building blocks for an anonymous, decentralized Peer-to-peer (P2P) overlay network were constructed and evaluated. Despite its potential, Fallang's prototype was limited by its need for substantial manual input and prerequisite knowledge among users for effective network creation. In this thesis, the focus is on exploring different approaches to bootstrapping networks, meaning how to facilitate the incorporation of new users into a network. An extensive overview of the frequently encountered bootstrapping challenges is compiled, and potential solutions are presented. These ideas are then applied in combination with our own to experiment with bootstrapping the small-scale peer-to-peer decentralized overlay network created on top of Tor.

Two autonomous bootstrapping mechanisms are implemented: One utilizing the recently deprecated InterPlanetary File System (IPFS) Publish/Subscribe (Pub/Sub) system, and the other leveraging Tor Onion services. Both mechanisms utilize public key signing algorithms for peer identification. They also provide publicly accessible communication interfaces for each peer as well as automate the establishment and termination of direct connections between network peers. The comprehensive evaluation based on criteria such as performance, reliability, design, security, and anonymity revealed that the Tor Onion service-based bootstrapper outperforms its IPFS counterpart in most aspects. This suggests its suitability for small-scale anonymity networks. The finding from this thesis has implications for the ease of deployment and efficiency of anonymous decentralized P2P networks.

Sammendrag

Denne masteroppgaven er en fortsettelse av det grunnarbeidet som ble gjort i den tidligere masteroppgaven av Fallang [1], hvor de fleste av de grunnleggende byggesteinene for et anonymt, desentralisert likemannsnettverk ble konstruert og evaluert. Til tross for sitt potensiale, var Fallangs prototype begrenset av behovet for en betydelig mengde manuelt arbeid og forhåndskunnskap blant brukerne for å opprette slike nettverk. I denne oppgaven fokuseres det på å utforske forskjellige tilnærminger til å "bootstrappe" nettverk, altså hvordan etablere og integrere nye brukere inn i et nettverk. Det ble laget en omfattende oversikt over utfordringene ved bootstrapping, og mye brukte løsninger er presentert. Disse ideene blir deretter brukt sammen med nye idéer for å eksperimentere med bootstrapping av Fallangs småskala desentraliserte likemannsnettverk som er laget på toppen av Tor nettverket.

Det har blitt utviklet to autonome bootstrapping-mekanismer: en som utnytter det nylig foreldede IPFS Publiser/Abonner-systemet, og den andre som bruker Tor sine "Onion tjenester". Begge mekanismene bruker offentlige nøkkelsignaturalgoritmer for identifikasjon av brukere og lager et offentlig tilgjengelige kommunikasjonsgrensesnitt for hver bruker, samt automatiserer etableringen og opphørelsen av direkte forbindelser mellom brukere i nettverket. Den omfattende evalueringen basert på kriterier som ytelse, pålitelighet, design, sikkerhet og anonymitet viste at den Tor Onion-tjenestebaserte bootstrapperen overgår sin IPFS-motpart i de fleste aspekter. Dette antyder at den er egnet for småskala anonymitetsnettverk. Funnene fra denne oppgaven har implikasjoner for utrullingen, etableringen og effektiviteten av anonyme desentraliserte likemannsnettverk.

Preface

This thesis was submitted to finalize my master's degree in Communication Technology and Digital security at the Norwegian University of Science and Technology. The thesis is a continuation of the pre-project conducted in the fall of 2022, and the master's thesis itself was completed in the summer of 2023.

I want to thank my supervisor, Lasse Øverlier, for his guidance during the pre-project and while writing this thesis. I also want to thank all the great people I have enjoyed getting to know and collaborating with during my four years at Studentmediene in Trondheim. In particular, I owe much to the IT department, which has made my time in Trondheim not only rewarding and educational, but also thoroughly enjoyable. Lastly, a heartfelt thanks go to my family, friends, and my better half, Astrid, for all the love and support they have given me during these years.

Contents

List of Acronyms

BRB	Bramble Rendezvous Protocol.
CID	Content Identifiers.
DA	Directory Authority.
DH	Diffie-Hellman.
DHCP	Dynamic Host Configuration Protocol.
DHT	Distributed Hash Table.
DNS	Domain Name System.
DoS	Denial of Service.
IoT	Internet of Things.
IP	Internet Protocol.
IPFS	InterPlanetary File System.
IRC	Internet Relay Chat.
ISP	Internet Service Provider.
LAN	Local Area Network.
NAT	Network Address Translation.
NKN	New Kind of Network.
P2P	Peer-to-peer.
PoW	Proof of Work.
Pub/Sub	Publish/Subscribe.

QOS	Quality of Service.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
VPN	Virtual Private Network.

Chapter 1

Introduction

In the realm of network communications, Peer-to-peer (P2P) overlay networks serve as integral frameworks. This enables the formation of networks independent from their underlying network topology and without centralized entities. Bootstrapping, which involves the initialization of networks, the discovery of participating peers within such networks, and the automatic establishment of direct communication links among them, is a challenge with no straightforward solutions for all networks.

To begin, a comprehensive review of the literature on existing approaches is undertaken, explaining how the process of bootstrapping in P2P networks and services has been proposed and is currently employed. This review aspires to gather a thorough understanding of the associated challenges, particularly in relation to network operations and security. Furthermore, it aims to shed light on existing solutions to these issues and those currently deployed.

Subsequently, two distinct bootstrapping mechanisms are designed and implemented specifically for an overlay network created on top of Tor. While one mechanism offers anonymity, the other does not. By leveraging established technologies such as Tor onion services and the InterPlanetary File System (IPFS)'s decentralized Publish/Subscribe (Pub/Sub) architecture, cryptographic identifiers of already participating peers in the network can be distributed. The same technologies facilitate publicly accessible communication interfaces tied to these identifiers for both scenarios. These mechanisms aim to empower users to establish connections among themselves seamlessly and autonomously in a decentralized manner.

Empirical measures of network performance and failure instances are ultimately collected and analyzed. This quantitative evaluation and discussion of the design and security attributes contribute to a better understanding of these bootstrapping mechanisms' overall efficiency and real-world applicability.

1.1 Keywords

Bootstrapping, decentralized, overlay networks, peer-to-peer, Tor, Onion services, IPFS, publish-subscribe, anonymity.

1.2 Motivation

Numerous anonymous P2P systems are already in existence, with Fallang's[1] prototype leading the way as the first to harness the Tor network as an anonymous transport. This facilitates a P2P network that can transport almost any type of traffic while providing anonymity. As proof of concept, their network is commendable; however, improved bootstrapping methods are one of the essential changes needed to transform this into a practical, user-centric anonymous service. Though functional, the prototype's bootstrapping mechanism requires users to pre-share their identifiers and consent to establish a connection within a brief time interval. A fresh agreed-upon time must be set in case of a failed connection. The complexity and inconvenience of these procedures render the system unattractive and burdensome for potential users.

In response to these challenges, the aim of this thesis is to devise an autonomous solution requiring minimal pre-shared information, easing the user experience while also being secure. Additionally, it sought to explore new bootstrapping strategies, considering that this area remains a pressing issue yet to be resolved within the realm of decentralized P2P overlay networks.

1.3 Objective

The main objective of this thesis is to explore techniques and improve upon the bootstrapping of the Tor overlay network. The steps will be to explore bootstrapping in decentralized overlay networks and then create an autonomous bootstrapping mechanism that makes establishing and joining the P2P network fast and easy, with the least amount of pre-known information. The bootstrapping mechanism does not have to be decentralized, but it would be preferable due to its potential beneficial properties. To solve these objectives, the following research questions are defined:

1.4 Research questions

Drawing parallels with the preceding project to this thesis [2], the research questions are constructed based on the objectives and are as follows:

RQ1 *What are the challenges of bootstrapping communications, and what are the most common approaches to overcome them?*

RQ2 *What techniques are most suited for bootstrapping an overlay network, e.g., built on top of Tor?*

1.5 Scope and contributions

This thesis mainly contributes to two parts of the research domain. The first is an overview of the most prevalent challenges with bootstrapping in existing computer networks, mostly focusing on P2P overlay networks. The overview includes bootstrapping of modern P2P networks as well as industry-established solutions. Some of these networks are not widely studied, and this will provide insights for further research on the topic.

The second major contribution of this thesis is the formulation of two unique autonomous bootstrapping mechanisms designed specifically for the Tor P2P overlay network. The first is utilizing an existing decentralized P2P overlay with Pub/Sub functionalities to provide a peer discovery and communication channel. We have not seen Pub/Sub systems used for this purpose in prior research or in other networks. The second mechanism expands upon previous bootstrapping mechanisms by applying them in combination with Tor Onion services and does not introduce new dependencies. However, these techniques are combined to solve the requirements that an autonomous bootstrapping mechanism in the Tor P2P overlay requires and evaluated in terms of performance, design, and security.

1.6 Limitations

Overlay networks and services that require some form of bootstrapping are replaced faster than the research community around it can investigate them. These systems are not always open-source; if they are, their documentation may be lacking or dispersed around on Wiki pages, forum posts, and software repositories. Additionally, it might also be outdated because of its rapid development schedule. This makes the investigation into these systems hard and forces us to rely on potential non-credible sources. Whenever possible, we try to cross-check the statements across multiple sources.

In Fallang's [1] initial design, the routing components between peers were omitted and simplified through the utilization of OpenVPN. We will maintain this simplified approach as the implementation of these routing aspects would demand significant additional effort while not directly enhancing the network's bootstrapping process.

1.7 Thesis outline

Chapter 1: Provides an introduction to the research domain and problem. Includes objectives, research questions, motivation, scope, and limitations.

Chapter 2: Introduces necessary background information for the thesis, such as bootstrapping, overlay networks, Tor, and publish-subscribe.

Chapter 3: Details the research methodology, justifying its adoption, explaining the literature review and software development approaches.

Chapter 4: Summarizes literature review findings on bootstrapping challenges and solutions.

Chapter 5: Outlines bootstrapping requirements and how the software is designed and implemented.

Chapter 6: Presents key results in the form of quantifiable performance metrics.

Chapter 7: Analyzes and discusses the results, architectural choices, and security aspects.

Chapter 8: Gives a conclusion that summarizes the study and its key findings.

Chapter 9: Suggests potential areas for future research based on the study's findings.

Chapter 2
Background

This chapter outlines the concepts and technologies explored in this thesis. The focus will be on overlay networks and bootstrapping, including a presentation of fundamental ideas necessary for understanding the design of bootstrapping mechanisms, which will be presented in later chapters.

The term "centralization" applies not to the underlying system architecture but rather to the management and control of said systems [3]. In contrast, a "decentralized" system is one in which control is not confined to a limited group of actors but rather distributed throughout the entire system. While "distributed" is often used interchangeably with "decentralized," it denotes a system where various components are located in separate physical locations [3].

2.1 Overlay networks

Overlay networks, or overlays, are computer networks built on top of another network [4]. The definition is broad, encompassing all networks that abstract away the underlying networks, also called underlays, and can route independently of the network they are built upon. Overlay networks are utilized to facilitate new features or services without requiring a complete network overhaul. These networks can provide new functionality and extend the underlay networks' feature sets with their own. Initially, the Internet was an overlay network, as it was built on top of the telephone network [4]. Today, overlay networks are in various application domains ranging from IoT to 5G mobile networks.

P2P overlay networks allow peers to locate each other through logical identifiers rather than IP addresses [5]. These networks offer several benefits over client-server systems; they do not have a single point of failure and are more resilient and scalable. These overlay networks can also provide users with better reliability, routing, and security. Historically, P2P overlays have been used for distributed storage, file sharing, and real-time data and communication streaming applications [5]. In the last decade, it has become substantially more used with the rise of decentralized technologies like blockchains and Web 3.0 [6].

P2P networks can be categorized into two distinct types of overlays: *structured* and *unstructured*. Galuba et al. [5] describe unstructured overlay as arbitrary network topology formed by peers often chosen randomly. Although these networks exhibit considerable resilience to failures, their operational efficiency and performance level are comparatively inferior to the structured alternatives. On the other hand, structured overlays employ more efficient routing strategies, and the selection of peer neighbors is often based on factors such as proximity and available bandwidth.

Distributed Hash Tables

Zhang et al. [7] explain Distributed Hash Table (DHT) as decentralized structured overlay networks that provide a distributed key-value store abstraction for large-scale distributed systems. DHTs are common usage for P2P networks [7]. They can be utilized to build complex distributed services, including overlay multicast, anycast, Content Delivery Networks (CDNs), and distributed Domain Name System (DNS). They also address practical problems like load balancing, distributed storage, search, and communication. Several DHT implementations share a basic set of atomic functions, including GET and PUT data operations. Similar to regular hash tables, data is stored in a key-value manner, where the keys are hashes built for efficient lookup. Unlike regular hash tables, these are distributed among all participating peers in a P2P network. However, the implementations of these DHTs differ in their routing structure, data storage, retrieval mechanisms, peer management, and the information each peer needs to know about.

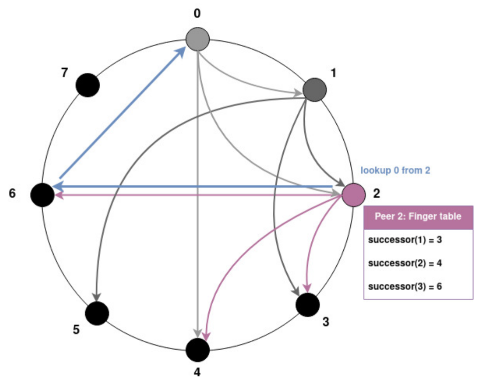

Figure 2.1: Chord DHT showing how to lookup the values at Peer0 from Peer2 going through Peer6 in a DHT with eight peers. It also shows the finger table assignments for the three first peers.

In the simple DHT implementation Chord, the DHT is visualized as a one-dimensional ring structure with consistent hashing. Each node has a unique node ID and is ordered in the ring by these [7]. Nodes maintain their own "finger table," in which they store a subset of all nodes in the Chord network. This is shown in Figure 2.1. Every node in the network retains the data with IDs falling within the range bounded by its own ID and the ID of its predecessor. Kademlia, the most popular P2P storage DHT alternative, introduces an XOR distance metric between peers allowing for more efficient lookup and storage by placing peers in different buckets. It has been adopted by many P2P applications, especially for file distribution services [7].

File distribution networks

P2P networks have gained recognition for their decentralized file-sharing capabilities, with numerous versions of such networks existing [8]. Among them, BitTorrent stands out as the most widely acknowledged and utilized variant. BitTorrent is a protocol aimed at efficiently distributing substantial volumes of data while circumventing the need for extensive server bandwidth resources. Through BitTorrent, an unstructured P2P overlay network is established for each torrent or collection of files being distributed, enabling a newly joined peer to simultaneously download the torrent from multiple peers. Upon acquiring a portion of the data, the freshly integrated peer can also actively distribute said data to other peers.

InterPlanetary File System (IPFS) is an open-sourced, P2P structured overlay network that provides free, decentralized, and reliable distributed storage of files [9]. It seeks to provide a globally distributed file system, connecting all computing devices to the same system of files. IPFS uses its Content Identifiers (CID) to uniquely identify data by a multihash checksum. This gives every version of a file a unique CID. If a file is too big, it is split into multiple CIDs linked together. CIDs are stored in a Kademlia-based DHT with reference to the peers it is stored at. IPFS has many use cases, and it is popular amongst decentralized Web3 applications for off-chain storage.

NKN

The New Kind of Network (NKN) represents a novel Web3 decentralized P2P overlay network constructed on top of the Internet. As described by the NKN whitepaper [10], it stands out as a new generation of self-incentivized blockchain network infrastructure characterized by its high scalability and self-evolving capacity. A proprietary cryptocurrency, NKN coin, is employed within the network utilizing a Proof of Work (PoW) mechanism, which avoids wasting resources and ensures network connectivity and data capacity. Furthermore, NKN deploys its own Chord DHT implementation hosted on the NKN network to establish its network topology.

The NKN network is partitioned into two types of participants, namely nodes, which undertake the task of data relaying, and clients that do not engage in such activity. Clients interact with the network by transmitting their data via nodes using their NKN address, which abstracts away the location of the device [11] and allows communication between clients, even behind Network Address Translation (NAT) restricted networks.

2.2 Tor

"The Onion Router," commonly known as Tor, is an overlay network designed to provide anonymity and to circumvent censorship of Transmission Control Protocol (TCP)-based internet services such as web browsing, instant messaging, and secure shell [12]. Tor is a distributed network intended to anonymize and protect users' privacy. It is inspired by the concept of onion routing from the mid-1990s, where traffic is routed through a network of proxy servers or relays, where each proxy only knows the previous and the next proxy in the chain. Before the traffic enters the network, it is encrypted multiple times, creating layers like the layers of an onion, hence the name [13]. Each server in the network removes one layer of encryption to reveal the address of the next server in the chain until the traffic reaches its final destination. Tor elaborates on this paradigm by incorporating circuits in which the initiating entity establishes session keys with each subsequent router of the circuit. This method serves to inhibit the decryption of previously transmitted data [12].

When a client and a server interact using an encrypted channel with Tor, only limited information can be gleaned by third parties capable of intercepting the traffic. These third parties may include the client's Internet Service Provider (ISP) or those with wiretapping abilities. They can only see that the sender is using Tor and nothing more. However, they will be unable to decipher the transmitted content or identify the recipient of the communication. It is possible for an observer who can view both the client and the destination website or the Tor exit node to correlate the timings of the client's traffic as it enters the Tor network and exits [14]. This illustrates that no perfect anonymity is possible but makes it significantly harder for someone to obtain information about the users when using an anonymous network like Tor.

In the present day, the Tor software has been made freely available and is subject to ongoing development and maintenance by the Tor Project, a non-profit organization. The project's guiding principle is encapsulated in the phrase, "Defend yourself against tracking and surveillance. Circumvent censorship." [14] The Tor network presently consists of thousands of servers dispersed worldwide and is operated by volunteers. The list of currently available relays is called the "Consensus" and is updated every hour. This update is carried out by Directory Authority (DA), a small list of trusted Tor project participants who have undergone vetting processes. Periodically, each

DA generates a view of all the available relays, which they sign and send to the other DAs. The DAs carry out a majority vote on the views received, and the resulting consensus is signed by all of them and used as the current consensus for the next hour [14]. Tor users request the updated consensus periodically from the DAs.

As stated, one of Tor's main goals is to circumvent censorship. When it works, Tor disguises all relayed internet traffic as Tor traffic, thus offering protection against traffic monitoring and surveillance [15]. To prevent censorship of Tor itself, Tor introduces the concepts of bridges and pluggable transports. They disguise Tor traffic by masking it to look like regular TLS traffic, making it difficult to identify with deep packet inspection methods. Tor hides its bridges' IPs, but techniques are accessible for finding these [15].

Onion services

The Tor Project also provides a means of anonymity for servers through Onion services, which were formerly known as hidden services. With Onion Services, TCP-based internet services, like Web servers, can use Tor to anonymize the location of their servers. These services are identified by their onion address, which for the current version (v3), comprises a 56-character long string that ends with ".onion." As depicted in Figure 2.2, these 56 characters are a Base32 representation of the onion service's self-generated Ed25519 public key, version number, and checksum. An onion address lookup is not resolved by the Domain Name System (DNS) but rather through Tor Hidden Service Descriptors. These descriptors are first made when the onion service is created and then updated daily. The onion service specifies randomly selected relays as introduction points and includes their public key in the descriptor. The descriptor is then signed with the onion service's Ed25519 private signing key and uploaded to a DHT in the Tor network for others to access. At this point, the Onion service is reachable by clients if they know the public key [16].

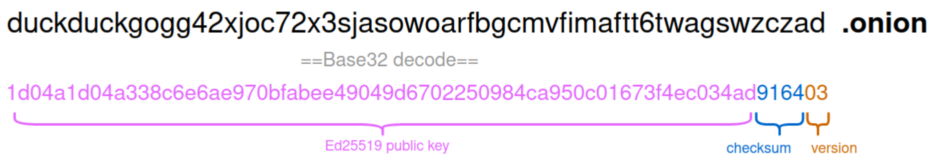
duckduckgogg42xjoc72x3sjasowoarfbgcmvfimaftt6twagswzczad .onion
==Base32 decode==
1d04a1d04a338c6e6ae970bfabee49049d6702250984ca950c01673f4ec034ad916403
Ed25519 public key
checksum
version

Figure 2.2: Building blocks of Tor v3 onion addresses

When a client intends to establish a connection with an Onion service, it initiates a request to Tor DHT by using the Onion address's public key as the lookup key. The DHT responds with a signed descriptor corresponding to the specified onion address. The client validates the descriptor's authenticity by utilizing the public key encoded within the onion address. This provides a guarantee of authentication from the server to the client. Subsequently, the client selects a random Tor relay to serve as its rendezvous point and initiates a circuit to it. The client then uses one of the onion service's introduction points to request the establishment of a Tor circuit at the rendezvous point, through which the client can access the onion service. This way, the client and the server can communicate anonymously without either being able to identify the other's network identities. Both the server and the client have a three-hop circuit to the rendezvous point, making the final established circuit consist of 6 relays [16].

2.3 A decentralized P2P overlay network built on top of Tor

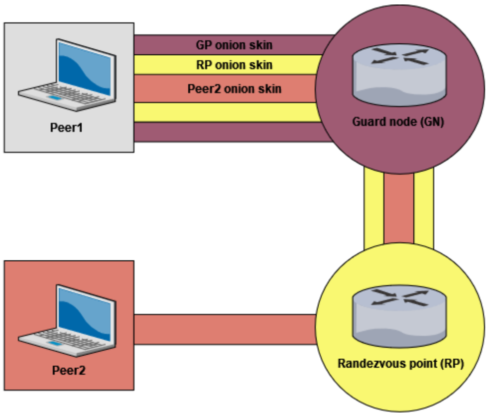

Figure 2.3: Illustrates the layers of encryption and the different nodes used in an established tunnel in Fallangs prototype. This illustration is based on Figure 4.8 in Fallangs thesis [1].

Fallang [1] developed a decentralized overlay network prototype that utilizes the Tor infrastructure as an underlay in his recent thesis. The network inherits Tor's anonymity properties, rendering it an anonymous, decentralized, peer-to-peer overlay network. The prototype enables two peers to establish a direct link via a series of Tor relay nodes. OpenVPN is implemented atop the Tor Socket to simplify routing and allow traffic other than TCP, which is the only transport Tor Sockets allow. During the connection setup, one peer assumes the role of a server, and the other acts as a client. The server creates the initial Tor circuit using a two-hop relay chain, with the client subsequently connecting through a single relay. The prototype is extendable to incorporate additional hops, potentially enhancing anonymity. Figure 2.3 illustrates the different layers of encryption used to mask the traffic in Fallang's prototype, as well as the two-hop relay chain from the server's perspective, Peer1, through the Guard Node (GN) and the Rendezvous Point (RP). A rudimentary, non-autonomous bootstrapping mechanism was also implemented, allowing peers to establish this connection, albeit requiring significant prior knowledge and manual work to connect. Coordination between users is essential for successful connections.

2.4 Bootstrapping

Although the term may have different meanings based on the context, the process of joining a computer network is commonly referred to as *bootstrapping*. In most instances, the process is straightforward, with the responsibility of facilitating the bootstrapping often delegated to servers and transport primitives. However, these servers' nature and specific roles depend on the network's properties and the abstraction layer on which they are built.

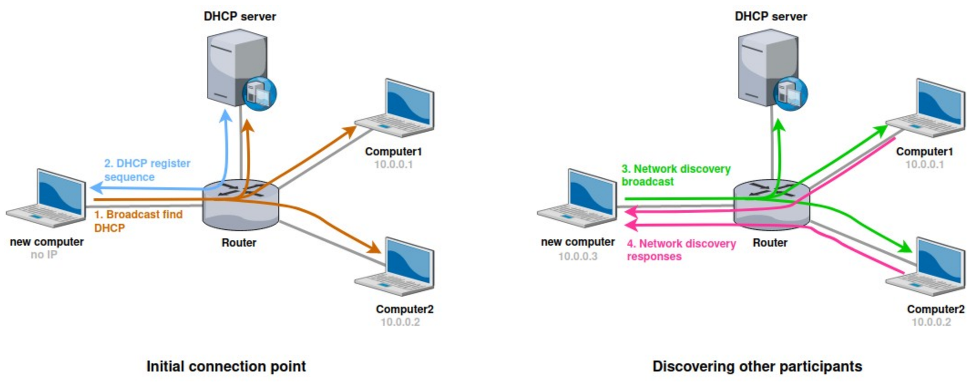

Figure 2.4: LAN Bootstrapping: Upon a new computer's integration into a LAN router, it engages the router to broadcast for and locate a DHCP server, initiating an exchange of messages that results in the acquisition of an IP address and network configurations. The new computer employs network discovery mechanisms such as Address Resolution Protocol (ARP) or multicast DNS (mDNS) for identifying other computers within the LAN.

For instance, when we consider Local Area Network (LAN) - among the simplest network setups - bootstrapping a new computer into the network is shared between the router and the Dynamic Host Configuration Protocol (DHCP) server. This is illustrated in Figure 2.4, where a new computer device will broadcast a request to join the network and request information from the current DHCP server. The router facilitates this message broadcasting. The DHCP server maps the client's unique Media Access Control (MAC) address to an unused IP address, leading to a series of messages between the new client and the DHCP server resulting in a successful network bootstrapping. Although this example is straightforward, it illustrates some necessary components for a successful bootstrapping mechanism, as there is a need for an initial connection point, a way of getting configuration information, and a way of discovering other participants on the network.

However, the task of bootstrapping is not always straightforward. This is particularly true for networks with decentralized components, such as in some P2P overlay networks. In P2P networks, bootstrapping is also referred to as *peer discovery.* And when it needs to be decentralized it is called the *bootstrapping problem.* According to Knoll et al. [17], a complete bootstrapping mechanism facilitates the integration of new peers into the network, even in situations with few or no existing peers. Knoll outlines five essential requirements that must be satisfied by a robust bootstrapping mechanism to qualify as decentralized. The first requirement, *Robustness against failure*, mandates that all components are decentralized and free of single points of failure. The second requirement, *Robustness against security appliances*, requires users to be able to communicate with each other and traverse firewalls and Network Address Translation (NAT). The third requirement, *Robustness against external interference*, necessitates complete decentralization of all components to prevent any authority from shutting down the network. The fourth requirement, *Efficiency*, ensures that the bootstrapping process is expeditious and involves the least amount of network traffic possible. Finally, the fifth requirement, *Scalability*, stipulates that the bootstrapping mechanism should not limit the number of users in the system. Doyle [18] suggests a sixth possible requirement: *Lawfulness and Ethical Use.* He argues that the bootstrapping protocol shouldn't violate any rules or policies. Furthermore, it should not misuse any existing technology, as such an approach would only offer quick but unsustainable, or 'dirty,' solutions to the bootstrapping problem.

While certain overlay networks are decentralized, their bootstrapping method need not invariably align with this decentralized approach. Rather, whether to use a centralized or decentralized bootstrapping mechanism usually hinges on the rationale behind the network's creators' adoption of a decentralized infrastructure. Should the motivation come from distrusting central authorities among the network's intended users, a decentralized bootstrapping approach would be the more favorable option. Additionally, the choice of a centralized bootstrapping method often rests upon considerations of operational costs, the network's scalability, robustness, and level of complexity. It is worth noting that while many decentralized networks utilize a semi-centralized bootstrapping approach, there are examples of decentralized approaches as well.

2.5 Efficient message-delivery mechanisms

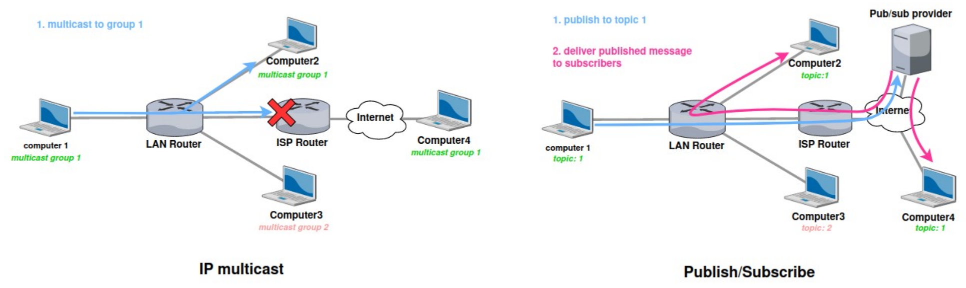

Figure 2.5: Contrast between IP Multicasting and Centralized Pub/Sub Systems: In the scenario of IP multicasting, every member of a common multicast group within a LAN is the recipient of packets addressed to the multicast destination. Notably, IP multicast tends to be obstructed at ISP routers. On the other hand, messages from a Pub/Sub system are disseminated to all participants, extending even beyond the boundaries of the LAN. However, the Pub/Sub system duplicates packets at the provider's end and dispatches them individually to each subscriber.

Multicast is a packet delivery design pattern for efficient one-to-many data transfer applications [19]. It allows the decoupling of the amount of information each network node needs to have from the network to send data to multiple devices. Multicasting can be divided into two broad categories based on the networking layer on which they operate. Network-layer multicast is the first, and because of its broad mentions in the literature, it will be rendered synonymous with the IP multicast protocol. IP multicast has been regarded as a great, best-effort, large-scale multi-point content delivery protocol which is bandwidth efficient [20]. IP multicast is, however, broadly limited to networks under single administrative control, such as LANs and enterprise networks. ISP usually stops multicast traffic in order to protect against unwanted traffic and reduce router load. This is shown in Figure 2.5. This severely reduces its applicability on the Internet. To combat these ISP restrictions and provide a similar feature set as IP multicast, multicast systems can be built on the application layer instead. Application-layer Multicast can be overlay-based, and instead of replicating data at network routers, it is replicated on end systems, also known as end-system multicast.

Publish/Subscribe (Pub/Sub) systems provide similar features as multicast. It's a well-established solution for facilitating decoupled and many-to-many messaging [21]. There exist numerous diverse implementations of the Pub/Sub model. Despite their varying nature, all such systems are designed to offer the ability to subscribe to specific topics. In practice, this means that whenever a message is published pertaining to a subscribed topic, the system endeavors to deliver this message to all parties that have subscribed to that particular topic. Other features message delivery guarantees or authorization. The Pub/Sub model has mostly been used in the Internet of Things (IoT) domain for applications such as real-time data collection and distributed messaging [21]. MQTT and DDS are commonly used Pub/Sub protocols for IoT applications [22]. MQTT uses centralized servers called brokers that manage topic subscriptions, receive published messages, and redistribute messages to the subscribers of the given topics. An example of a centralized Pub/Sub system is shown in Figure 2.5. DDS manages an overlay network to route its users' published messages to the subscribers. It does not have a broker, but it still upholds different Quality of Service (QOS) guarantees [22], like MQTT.

As described in Section 2.1, IPFS and NKN are examples of new decentralized P2P services. Although these services are structured to serve distinct functionalities, they both incorporate a Pub/Sub mechanism within their respective overlay networks. This Pub/Sub model, being entirely distributed and decentralized, circumvents the necessity of a centralized broker to facilitate the distribution of published messages to subscribed topics.

Reviewing the repository of the Go-language implementation of IPFS, known as Kubo [23], reveals that IPFS essentially functions as a wrapper around the *libp2p* [24] Pub/Sub functionality. Furthermore, IPFS employs two unique routing strategies, each of which creates a distinct P2P overlay network corresponding to each subscribed topic. The part that distinguishes these strategies lies in their approaches toward routing between peers associated with a specific topic. The initial strategy, namely FloodSub, espouses the naive approach of flooding the entire network with published messages. However, the most recent and now default approach, referred to as GossipSub, leverages a gossip-based method for routing messages, adding scalability to their Pub/Sub model. These topic-focused overlay networks necessitate the existence of peer discovery mechanisms to identify other peers within the network. In this context, the IPFS Pub/Sub employs diverse strategies, ranging from bootstrapping servers to Local Area Network (LAN) multicasting.

In contrast, NKN employs a distinctly different strategy in shaping its Pub/Sub architecture. Its implementation capitalizes on its proprietary blockchain to store subscribed topics, thereby making them retrievable and searchable by other entities. An incidental result of this approach is the requirement for users to utilize a blockchain wallet to authenticate and store subscriptions in transactions. In this case, the NKN network assumes responsibility for the routing and subsequently serves as a highly efficient system for message propagation with its milliseconds' end-to-end latency [25].

Chapter 3
Methodology

This chapter outlines the strategy for answering the two research questions presented in Chapter 1.4 and discusses and justifies the choice of methodology.

The thesis will answer the questions sequentially, as the answer to Research Question 2 (RQ2) depends on Research Question 1 (RQ1). The justification for this is that RQ1 will provide insight into some of the issues that will later be encountered when attempting to tackle RQ2. RQ2 will, however, receive most of the time and focus since it is the most time-consuming part of the thesis as it involves development. It will bring more insight and creativity to the field's ongoing study.

When addressing RQ1, an overview will be created of the problems associated with bootstrapping different P2P networks. These problems will be considered while designing, implementing, and evaluating different approaches on top of Fallang's prototype in RQ2.

3.1 Overview of the state of Bootstrapping

To address RQ1, an overview of the leading problems associated with bootstrapping in both classical and new P2P networks will be presented. In recent years, research institutions and the industry have investigated many of the associated challenges, which can be employed to improve the insight into the presented research questions. At the time of starting the thesis, a list of relevant literature is already acquired. A literature review will be conducted, and the following steps will be taken:

1. Utilize standard resources and search engines like the university's library's search portal, Google Scholar, connected papers, and the normal Google search engine.

2. Use relevant keywords such as "bootstrapping," "decentralized," "peer-to-peer," "overlay network," "ad-hoc networking," "distributed applications," "pervasive computing," "peer discovery," and "distributed computing" in searches.

3. Create a list of existing P2P applications and overlay networks, examine their approaches to bootstrapping, see if they approach bootstrapping in novel ways, and pick out a few interesting ones. The initial selection criteria are if they have either a lot of current practical usages or are mentioned a lot in the literature. Other services that include special properties like anonymity will also be looked at.

4. Organize the different approaches of bootstrapping based on the challenges they are trying to overcome, and if they are networking or security and privacy related.

This method is similar to the one proposed method in the project leading up to the thesis [2]. At the end of the literature review, the overview will serve as the basis for tackling the second research question. The literature review is presented in Chapter 4 as it serves as a standalone part of both the background material and the results of this thesis.

3.2 Exploring and implementing bootstrapping mechanisms

To address RQ2, an exploratory approach will be undertaken. The goal is to enhance the decentralized Tor overlay prototype by Fallang[1] with the creation of an autonomous bootstrapping mechanism that prioritizes ease of use and security. This overlay network provides the unique property of transport anonymity using the Tor Network as a substrate for the overlay. Its current non-autonomous bootstrapping mechanism requires that two peers know their peer's self-given id and try to connect relatively quickly. To create a connection between two peers, one peer must act as a server and the other as a client. Additionally, there is a requirement for the peer acting as a server to be the first to establish their part of the tunnel. The pre-shared information is used to choose which Tor Rendezvous point they intend to use to start a relaying connection between them. This mechanism will serve as the foundation, with further development branching out in various directions. The aim is to discover a superior method of deriving a shared secret to facilitate rendezvous selection between peers. Ultimately, the intention is to devise a bootstrapping mechanism that either inherits the transport anonymity of the network or possesses the capability to provide it.

Development methodology

Given that the existing Tor overlay prototype is implemented in Python, it has been determined that continuing with this established codebase would be beneficial. This decision is partially due to Python's renowned capacity for rapid development and facilitation of prototyping [26], which aligns seamlessly with the project's goals. However, it is also acknowledged that some alterations to the existing code are required. Undertaking these changes in a different language would introduce additional overhead and strengthen the choice of persisting with Python.

The planned development methodology encompasses a fusion of the waterfall and the iterative software development models [27]. This results in a development process that involves requirement gathering, while also allowing for adjustments whenever the requirements are found to be unsuitable. By merging elements from both models, the intention is to achieve enhanced flexibility and adaptability while maintaining the importance of upfront planning and requirement gathering. The decision to adopt this particular approach is largely guided by the team size, which in this case, includes just the author. More intricate development methodologies like Agile are typically designed for collaborative teams. Therefore, in this context, these complex strategies are deemed unnecessary. Furthermore, the intention to explore the domain through the creation of prototypes, and the acknowledgment of the potential gaps in our initial understanding of the problem, justifies the incorporation of iterative aspects into the methodology.

The initial step involves the identification of the prospective user base and the elicitation of the software's requirements. Simultaneously, outlining the restrictions and limitations arising from reliance on chosen software platforms will be necessary. This aligns well with the first steps of a waterfall model.

The ideation phase will pivot around these established requirements and restrictions, promoting iterative brainstorming to ensure all proposed concepts align with these parameters. The intention is to utilize sequence diagrams to facilitate the communication of our proposed software design and improve our understanding in a visual manner. These diagrams will undergo iterative refinement to best reflect the intended design.

The methodology incorporates the use of state machines and components in the design and development of the software. State machines will be used to clarify system behavior by representing different components as discrete states and transitions. This approach facilitates decoupling an event's originator from its responder [28], thereby enabling different components to transmit events amongst themselves in a well-defined manner. Utilizing state machines can effectively streamline the debugging process, reduce the design's complexity, and enhance our comprehension of communicating concurrent software components [28]. Another important attribute of state machines is their contribution to the maintenance of documentation, as the plan is to update the diagrams representing them prior to their implementation. This enables a visual design process and ensures consistency between documentation and implementation.

Once a satisfactory level of understanding and confidence in the proposed design is achieved, the software development phase will commence. Nevertheless, software development projects often necessitate revisions in response to emerging insights and additional requirements. Therefore, there is an anticipation of revising diagrams and adapting the software to these changing circumstances, ensuring that the final prototypes meet the current requirements. The final requirements and designs will be presented in Chapter 5.

Assessment

To evaluate the effectiveness of the bootstrapping mechanisms, there is an intent to analyze their performance with regard to speed as the time to establish a network and the time to join an established network. The occurrences of connection attempts and failures will also be measured. These quantifiable metrics offer clear data points for measurements and facilitate meaningful comparisons. The outcomes of these empirical investigations will be presented in Chapter 6. However, the main results are the bootstrapping mechanism themselves; evaluating these and resolving the second research question necessitates a more thorough analysis. Chapter 7, undertakes an analysis and comparative assessment of these mechanisms. Their respective security, anonymity, and usability attributes will be considered, with the overview from RQ1 serving as guidance.

Chapter 4

The challenges of bootstrapping

This chapter aims to provide an overview of the main challenges involved in the bootstrapping of pre-existing computer networks in response to Research Question 1. Considering their extensive coverage in prior research, the focus predominantly rests on P2P overlay networks. The challenges looked at are grouped into two basic categories: Networking, security and anonymity. For each challenge, the issue is clarified, and theoretical approaches along with established solutions in existing networks are presented if they exist.

4.1 Networking

Bootstrapping computer networks incorporates challenges that are strictly network related. This section presents a set of networking problems and ways of solving them.

4.1.1 Finding the first peer

One of the foremost challenges associated with bootstrapping arises in scenarios where the network is decentralized. Frequently, there is a desire for the associated bootstrapping process to be decentralized as well. As elaborated upon in Chapter 2, electing to employ a decentralized architecture frequently leads to the conundrum of finding the first peer, referred to as the bootstrapping problem [29]. This challenge asserts the following requirements; that the network is of a peer-to-peer nature and that all peers can bootstrap further peers, provided they are both known and reachable. This problem has been researched extensively, as there is no solution that works for all networks.

Bootstrapping servers, nodes, and DNS seeds are the predominately used solution to the problem of finding the first peer. This refers to the different cases where the network's creators host a set of bootstrapping servers and, in some cases, allow anyone or trusted parties to host their own. Hence, it's up to the users to configure the bootstrapping servers they want to use. Still, the default set of servers is usually highly reliable servers configured and maintained by the network creators. It also refers to cases where the servers are changed out with established long-lived peers in the network. This is the case for the popular distributed storage and file-sharing applications IPFS [30], and BitTorrent [31]. These networks use a Kademlia DHT and require peer bootstrapping, meaning new users must be incorporated into the network by other peers. Their client software uses hardcoded DNS seeds and highly available IP addresses of established peers as the first-time bootstrapping mechanisms. The DNS seeds let the developers hard-code a set of domains, thus extracting the node lookup into DNS. This makes it easier for them to replace the bootstrapping servers without making the network's users update their software when they fail or get new identifiers. Hard-coded IP addresses are used as a fallback for these DNS seeds in the case of DNS failure. When queried, the DNS seeds provide a set of bootstrapping nodes or servers that will either bootstrap them into the network or provide active peers in the network upon request.

NKN implements a similar bootstrapping approach. However, NKN's architecture utilizes a Chord DHT [11] and incorporates distinct bootstrapping mechanisms for their relaying nodes and their clients. The bootstrapping process for nodes is more intricate than that of clients, but both rely on bootstrapping servers to get the initial list of nodes in the network. NKN nodes must maintain lists of their successors, predecessors, neighbors, and connected clients. The former three are first populated in the bootstrapping procedure and continually updated while in the network. This is accomplished by initially requesting a list of other nodes from a bootstrapping server and subsequently joining the DHT by contacting either of these. In contrast, clients request a list of NKN relaying nodes from the bootstrapping server and connect to another node assigned by one of the retrieved nodes.

IPFS and NKN links a user's verifiable public key to their node identifier to provide authenticity and integrity for their users. This also helps remove the need to know a peer's IP and their physical location. Both allow multiple ways of reaching the same peer by their id, for example, when a user has multiple devices. IPFS does this through their "multiaddr" address scheme, and NKN uses their NKN addresses which can act similarly to subdomains.

The P2P networks that underlie most of the current digital cryptocurrencies also share the use of distributed bootstrapping nodes. Loe and Quaglia [6] surveyed the bootstrapping mechanisms for all of the mineable top 100 cryptocurrencies in 2019. Their findings show that all but one of these P2P networks use either DNS seeds, hard-coded bootstrapping servers, or a combination of the two as their main peer-discovery mechanisms. Multiple implementations of P2P networks exist, whereas some implementations utilize DHT to maintain their networks. This holds for the two most popular cryptocurrencies, Bitcoin and Ethereum, built upon their P2P networks named "the Bitcoin network" [32] and "devp2p" [33] respectively. Like IPFS and BitTorrent, Ethereum's devp2p protocol, use a Kademlia DHT to create and maintain its network topology, using a proprietary addressing scheme. On the other hand, the Bitcoin network uses no form of DHT for its P2P network and relies on full nodes having a list of most of the available peers. These lists are populated with Gossip protocols, which facilitate the efficient distribution of information across the network by allowing peers to "gossip" or share information with their neighbors in a decentralized way[34].

Caching previously known peers is a bootstrapping technique employed in most large-scale P2P networks. Each client must locally cache a database of peers retrieved from another bootstrapping mechanism or after joining the network. It cannot work as a standalone bootstrapping mechanism [18] and has to be used in addition to other methods presented in this chapter. The latency of the bootstrapping process can be substantially decreased when the cache is retrieved recently, diminishing the load burden on other bootstrapping mechanisms. Noteworthy, the approach of caching recently known peers is highly scalable, but it may not yield optimal results in small-scale P2P networks, as outlined by Doyle [18] and Knoll [17]. Furthermore, the duration since the last update of the peer cache is proportional to the likelihood of the peers in the cache being offline. Should the cache become entirely stale, signifying that there are no active peers within the network, its functionality would be compromised and deemed completely ineffective. Furthermore, this scenario would reduce the efficiency of the bootstrapping process, even when compared to instances where no cache system has been implemented. In stale cache scenarios, the user must fall back to other bootstrapping strategies.

The caching approach is great in cases with temporary disconnections from a network, letting the disconnected user connect to the network without going through another bootstrapping mechanism. [18]

Using external services for bootstrapping is frequently suggested in the literature. The external services are usually decentralized or distributed. Examples of these are using Internet Relay Chat (IRC) [29] and DNS [17] [35], or even recently, blockchain technologies [36] [37]. Knoll, Wacker, Scheile, and Weis [17] introduced the concept of using IRC for bootstrapping in 2007, and it got was utilized as one of the peer discovery mechanisms of the early Bitcoin network [38]. It gave the network a decentralized method of bootstrapping when the pool of static IPs in the network where insufficient and changed frequently. With IRC bootstrapping, each peer encoded their own IP into an IRC nickname, then they randomly would join an IRC channel named between *#Bitcoin00* and *#Bitcoin99*, issued an IRC WHO command, and decoded the IP addresses from the nicknames listed. The addresses found would be other peers already bootstrapped and could be used to join the network. The feature was a temporary solution and was removed in 2013 as part of Bitcoin's peer-discovery mechanisms [38]. Many alternative cryptocurrencies are forks of Bitcoin and may still use this method as one of their bootstrapping approaches.

Bitcoin itself has been proposed as an external service usable for bootstrapping. Matzutt et al. [36] created a prototype for an anonymous bootstrapping mechanism named AnonBoot, by encoding periodic peer advertisements into the 80 bytes *OP_RETURN* field of Bitcoin transactions. The method is not restricted to Bitcoin and works on all blockchain technologies that enable arbitrary data storage in its transactions. New peers need to solve a small PoW puzzle in order to advertise themselves on the Bitcoin blockchain. Similarly, a bootstrapping mechanism using Ethereum Smart Contracts where proposed by Scutz et al. [37] in order to join a trackless DHT-based BitTorrent network. Although using blockchains for bootstrapping seems promising, neither of these approaches is used in currently deployed networks.

A problem with using external services as bootstrapping mechanisms is that the network using them will depend on that service's reliability. Additionally, if a peer needs to be bootstrapped into that external service, the problem is just shifted to let the external service deal with it. For IRC bootstrapping, one relies on the IRC servers continuing to operate and might be a central point of failure. When the network grows, the bootstrapping mechanism will impose a greater load on the IRC servers, which might not align with the IRC servers' owners' intended purpose for that service.

Using broadcast or multicast primitives such as IP multicast for bootstrapping is a well-researched topic. As described in Chapter 2, IP multicast is not usable on the Internet as it tends to be blocked on the ISP level. However, IP multicast can locate peer nodes in local networks participating in the same overlay network [39]. This requires all the participants in the overlay to join the multicast group. New nodes or peers can query the multicast group for the IP addresses of the bootstrapped participants [40]. Using multicast in such a way could impose a height amount of traffic in the network, and the multicast search should be performed using techniques like expanding ring search [40]. Like peer cache bootstrapping mechanism approaches, multicast usually needs to be used in addition to other approaches.

IPFS uses multicast DNS (mDNS) as one of its bootstrapping mechanisms in LANs. It provides an efficient means of finding other peers, even without being connected to the internet or relying on IPFS's bootstrapping nodes [41]. Similarly, the P2P dataset synchronization protocol *Dat* previously used mDNS as one of its bootstrapping mechanisms, or source discovery as they called it [42]. It also only worked on LANs.

Probing IP addresses refers to a collection of different methods proposed in the literature. These include random scanning of IPv4 addresses aiming at finding a peer in the network and using statics to improve upon these scans. The bigger the network, the greater the chance of finding a peer. Loe et al. [6] used 10 GBit network cards to scan the entire IPv4 address space with ZMap to bootstrap blockchain networks. Even though this method requires no centralized resources and is inherently censorship-resistant, they conclude it is not a realistic method of bootstrapping due to the high bandwidth requirements and low success rate. Although they found hundreds of IPs using the same ports as the blockchain P2P networks they surveyed, none successfully bootstrapped them into the networks. Ethereum-based P2P networks were not compatible with this search as they use dynamically allocated ports, and they were thus unable to consistently identify whether IPs belonged to peers in these networks.

Other IP probing mechanisms like random address probing [43] and locality-aware address probing [40] have been proposed in the literature. To the best of our knowledge, none of these mechanisms have seen practical usage in current P2P overlay networks.

4.1.2 Churn

Churn refers to the dynamics of peer participation within P2P networks. It specifically refers to the impact that the rapid influx and departure of numerous peers within a limited time frame have on the network [44]. This phenomenon is primarily an architectural concern inherent to P2P systems, and its implications extend beyond the challenges in the bootstrapping of these networks. Consequently, churn is addressed in this context due to its influence on the selection of bootstrapping mechanisms and its profound implications for the reliability and scalability of such networks. Churn predominantly poses difficulties in large-scale P2P networks, whereas its effects on smaller-scale networks are more manageable. Within the realm of bootstrapping, churn introduces various issues, including the hindrance of peer discovery due to the potential absence of previously known peers within the network [40]. Moreover, it frequently disrupts the network's topology, thereby diminishing the efficiency of routing and resource allocation, necessitating potential resolution within the bootstrapping mechanism itself. Furthermore, churn impacts the scalability of the network, demanding that the bootstrapping mechanism possess the capacity to accommodate a substantial influx of new users and maintain up-to-date lists of active peers and their identifiers within the network. According to Stutzback and Rejaie [44], churn "must be taken into account in both the design and evaluation of any P2P application."

Caching long-lived peers is an approach by Stutzback and Rejaie [44] that aims to enable the successful bootstrapping of all peers within a large-scale network without relying on centralized addresses while dealing with the peer discovery problem introduced by churn. Although it is merely a caching strategy, it can effectively reduce the burden on other bootstrapping mechanisms after the initial successful bootstrap. The concept is relatively straightforward: each peer maintains a sufficiently large cache of long-lived peers. Through their analysis of various large networks, the researchers observed that approximately 20-30% of peers exhibit an uptime exceeding one day. Consequently, these long-lived peers should be given priority in the caching process.

4.1.3 Small networks

Wolinsky et al. [45] discusses the bootstrapping challenges of small-scale overlay networks. Because of these networks' limited set of peers, all bootstrapping techniques used in large-scale P2P networks might be ineffective for these networks. Small networks usually have two problems; the creators lack competency in maintaining reliable dedicated bootstrapping services, and the operation costs of these services may get too high as the number of users increases [45]. These networks tend to have no existing peers in the network, which is a scenario large-scale P2P networks do not have to worry about.

Using existing services such as public overlays is a method that is both discussed in the literature and used in small-scale networks. Wolinsky et al. [45] propose the Extensible Messaging and Presence Protocol (XMPP), an open standard distributed chat protocol, as one of the potential existing distributed services exploitable for the purpose of bootstrapping small networks.

This method may also serve as an intermediate approach, facilitating progression from small-scale to medium-scale networks. This was exemplified by Bitcoin's use of an IRC-bootstrapping technique. Our belief is that this strategy was designed to function solely as a bootstrapping mechanism during the network's early stage when there were insufficient consistently active, long-term peers within the network.

4.1.4 Hidden and unreachable peers

Despite the increase in the number of internet-connected devices employing IPv6, a significant portion of these devices remains inaccessible directly through the Internet due to the utilization of NAT or firewall restrictions. Consequently, the utilization of certain P2P networks is rendered challenging, as devices situated behind NAT are restricted to establishing outgoing connections and cannot be contacted directly unless they initiate the conversation. While this limitation tends not to pose a problem for large-scale P2P networks wherein a new peer will have a sizeable pool of contactable peers, certain networks still need to facilitate the bootstrapping of peers. This is exemplified with anonymity-providing networks discussed in 4.2.1. This issue exhibits a substantial correlation to the challenge discussed in Section 4.1.3 as it predominantly manifests itself within such networks. Hence, the first proposed approach is employed to address both problems.

Using existing services such as public overlays is once again used as an approach for a bootstrapping problem. It can be used as a means of bootstrapping unreachable peers into networks. Wolinsky et al. [45] present a set of three essential properties that an overlay network must possess to serve as a viable mechanism for bootstrapping such networks. Firstly, the public overlay used must incorporate a mechanism for obtaining global IP or peer identifiers, referred to as *Reflection*. Secondly, the overlay should enable users to exchange arbitrary data, even in the absence of a direct IP communication channel, a property termed *Relaying*. Lastly, the overlay must be used to identify peers interested in the same P2P service, known as *Rendezvous*. As an illustration, they identified which existing networks that provide all of these properties, and as mentioned in Section 4.1.3, they proposed XMPP as an external service that fulfills these requirements. Each user in XMPP has a unique identifier in a similar form to email addresses; "username@domain." This gives XMPP the reflection property, decoupling their physical location, via their IP, from their username. They were able to add the relaying property by extending the XMPP protocol, and the rendezvous property was fulfilled by peers advertising their P2P service usage in a hashed resource identifier accessible to all users.

The utilization of IRC as a means of bootstrapping in the context of Bitcoin was addressed in Sections 4.1.1 and 4.1.3. While the original intent of Bitcoin's IRC-bootstrapper design did not specifically cater to the issue of unreachable nodes, it possesses the necessary capabilities to fulfill the three essential properties required for this purpose. The IRC nicknames can be employed to encode users' IP addresses, as exemplified in the Bitcoin IRC bootstrapper, thereby satisfying the reflection property. Furthermore, the rendezvous property is ensured through the utilization of the IRC WHO command, which enables the listing of all users connected to the channel, thus facilitating the advertisement of their identifier. Lastly, users can transmit arbitrary data through the IRC channel in the form of text, thereby satisfying the relaying requirement.

Relaying data through other peers or nodes is a method employed by some networks to address the unreachable peer problem. NKN assigns joining peers to their decentralized and distributed relaying nodes which will handle the routing of data between peers and circumvent the NAT problems [10].

4.2 Security and privacy

Security and privacy are crucial considerations in bootstrapping computer networks due to their critical role in protecting information integrity, confidentiality, and availability. Due to their decentralized architectures, the P2P model is also exposed to these threats. This section presents some common attacks and privacy-related problems within the context of bootstrapping.

4.2.1 Anonymity

Privacy has become a bigger part of people's digital life in recent years. Users want to be able to control who has access to their data. For differing reasons, people do not want to be identifiable by any means. Although most people believe that complete anonymity is infeasible on the Internet [46], some networks and services promise varying levels of anonymity for their users. We advocate the notation that if the bootstrapping mechanisms in anonymity networks fail to ensure anonymity, it poses a considerable risk to users. This is because the data used to join the network could be used to correlate and identify individuals, thus compromising their anonymity. Therefore, we emphasize the crucial role of these bootstrapping mechanisms in establishing and maintaining robust anonymity within such networks.

Using an anonymous transport like Tor is the identified main method capable of providing autonomous and anonymous bootstrapping for existing networks. Many cryptocurrencies allow proxying with Tor to bootstrap into their blockchain P2P networks [6]. However, doing so for these networks is usually not recommended because of the vast amount of data needed to download their blockchains. This can result in worse Tor performance for other Tor applications. Besides its relatively low level of bandwidth, this makes Tor impractical for bootstrapping most blockchain networks. Research shows that bootstrapping cryptocurrencies with Tor may not provide the levels of anonymity one would desire [6]

The Verge cryptocurrency is an example of a blockchain capable of utilizing both Tor and I2P to provide anonymity for their users. Tor is used as an IP obfuscation tool for Verge's end users [47]. Additionally, it is the default underlying transport for contacting bootstrapping servers hosted with onion services [6].

There are multiple communication protocols and services promising anonymity for their users. The Briar project [48] is an entirely decentralized, secure messaging service. It can use a variety of networking transports to create ad-hoc mesh networks in infrastructure-less environments, such as during crises. Furthermore, it can protect its users from surveillance on top of the internet. For the latter, Briar outlines their Bramble Rendezvous Protocol (BRB) [49] as a Tor-enabled bootstrapping protocol for its P2P network. Using BRP, two peers with previously exchanged X25519 public keys can connect to each other using their peers' public keys and their own private keys in a Diffie-Hellman (DH) exchange to derive a shared secret among them. Using the shared secret, they derive a rendezvous key using a Key Derivation Function (KDF) on their derived shared secret. Both peers generate ED25519 key seeds by first using the KDF on the rendezvous key and then piping it into a stream cipher. This results in both peers having their own set of ED25519 keys which they use to set up Onion services. These Onion services act as anonymous public interfaces for peers, letting them communicate without others knowing.

Speek Secure Messaging [50] is a more secure and anonymity-focused messaging application than the Briar project. Although they provide much the same functionality, Speek only addresses censorship resistance and anonymity compared to Briar's additional ad-hoc mesh networking focus. Speek uses Tor Onion services to provide an instant messaging system between its users' contacts. It provides an end-to-end encrypted communication channel between two parties that want to talk. Like Briar, it uses Tor Onion services as anonymous, publicly available user interfaces. Speek is P2P in the sense that connections are made directly to their recipient and routed through the Tor network without going through any other peers or servers. For the bootstrapping mechanism in this service, users must know the Speek id for the users they would like to establish a connection and communication channel. These Speek ids are shared through other means without them telling you how.

4.2.2 Censorship

Most nations impose some form of restriction or censorship on their citizens' internet traffic, thereby inhibiting the usage of certain applications deemed undesirable by those in power. This practice is not confined solely to nations with authoritarian regimes [51]. Cryptocurrencies are examples of P2P applications that are being censored by certain countries [6]. Censoring the bootstrapping mechanism can effectively censor the whole network, as new users are unable to join.

Utilizing Tor as a transport is, similarly to providing anonymity, a prevalent strategy for circumventing censorship during the bootstrapping stages of P2P applications. The cryptocurrency bootstrapping survey by Loe and Quaglia [6] outlines the adoption of Tor as one of two censorship-mitigated bootstrapping techniques deployed in current cryptocurrency networks. Given that some of the internet traffic is uncensored, all P2P traffic assumes the appearance of Tor traffic by employing Tor, effectively evading the censorship imposed on such P2P networks. However, as described in Chapter 2.2, this method becomes ineffective if the Tor network becomes subject to censorship. Pluggable transports and Tor Bridges can be used to obfuscate and disguise Tor traffic as regular TLS traffic [6]. This poses yet another problem: Discovering and configuring these private Tor bridges. The same predicament arises if anyone, including the authorities responsible for the censorship, can easily locate the private Tor bridges. The extent to which these authorities are willing to enforce their censorship may vary, and the finding and censoring of these Tor bridges might be considered too much work for some of them, rendering it a viable but not universally accessible method for users wanting to use these cryptocurrencies.

Out-of-band peer id exchange is a method many P2P networks support. The idea is that users would exchange peer ids physically or in other communication channels. As long as communication with that peer is not inherently censored, the new user could be bootstrapped by contacting that peer. This is censorship-resistant but requires manual work. The other peer must be available at the time of joining the network, and their IP or contact info must not change. This is frequently used for bootstrapping many censorship-resisting networks and services like Briar [49], Speek [50], Freenet [52] and the Tor overlay network by Fallang [1]. In its application design, Briar integrates the use of Quick Response (QR) codes, which users can physically scan to add peers to their respective contact lists [48].

IP scanning methods stand as the only genuinely censorship-resistant autonomous bootstrapping techniques identified. However, as described in Section 4.1.1, these techniques are not a suitable option for all P2P networks due to some networks utilizing dynamically created ports for their peers, making it extremely time-consuming to find other peers. Furthermore, a network needs to be of sufficient size to be able to realistically find other peers.

4.2.3 Sybil-attacks

Sybil attacks are a type of attack on P2P overlay networks where malicious users generate a vast number of node identifiers to get a disproportionally large influence on the network or to disrupt the availability and integrity of said network [53]. It is strictly not just a challenge associated with just bootstrapping. Still, the problem arises in the bootstrapping mechanisms when there are no barriers to joining the network, and assignments of node identifiers are done in a decentralized fashion by the users themselves. This allows malicious peers to masquerade themselves as multiple different peers. It is considered one of the most challenging attacks to prevent in structured P2P overlay networks [53], and could lead to network partitioning, Denial of Service (DoS) attacks, unfair resource allocation, and removing data redundancy of distributed file storage. In P2P networks utilizing consensus vote mechanisms, like blockchain technologies, these attacks threaten the validity of the consensus's end result [54].

Resource testing techniques aim to reduce the presence of fraudulent nodes owned by malicious actors. These techniques involve tasks that impose computational demands on individual nodes within a specific time frame and evaluate their storage and network capabilities. By testing the resources of these nodes and comparing them to what would be expected if they were independent, the goal is to detect and mitigate the influence of malicious actors [55]. Numerous proposals for these techniques are suggested in the literature [56]. It has been recommended as a minimal defense strategy against Sybil attacks to reduce the risk and consequence instead of trying to remove the problem [54].

Analysis of social relationships between peers in a network is a form of Sybil detection utilized in anonymous and censorship-resistant systems [57]. In centralized anonymous communication systems and decentralized P2P overlays, the risk of Sybil attacks can greatly be mitigated by only forming communication channels with users they trust in the real world. The id of their anonymous identity must be shared out-of-band either in a communication channel they trust or physically. Many of these services only allow new users by invitation. Sybil identities can be found in these systems by analyzing the social graph of their identity and comparing it to the graphs of known honest nodes. The malicious identities will likely have very few connections to honest nodes or none at all. This can be used to detect potential Sybil nodes.

Payment for registration is an approach that aims to mitigate Sybil-attacks in large-scale networks. The concept behind this approach is that in order for a malicious actor to gain control over a significant portion of the network, they would have to incur a substantial financial cost. By imposing a payment required for each registration or identity in the network, legitimate participants in the network would be willing to pay the registration fee as they have genuine reasons to join and contribute. However, for attackers who seek to create numerous fake identities to manipulate the network, the cost of registration would become prohibitively expensive if they want to control a significant portion of the network. Thus, it acts as a disincentive for attackers by increasing the cost and effort required to launch a successful Sybil attack. This approach is, however, not suited for small to medium-sized networks as the payment amount would not be large enough to disincentivize malicious actors to pay for a proportionally large amount of identities [58].

4.2.4 Denial of Service

Denial of Service attacks are problems most publicly available computer networks must consider. These attacks refer to various ways that malicious actors can deny access to networks ranging in severity from denying a single user to all users of the system. Network layer DoS attacks typically involve flooding their victims with network traffic, thus overwhelming the victim's resource capacity [59]. In application-level DoS attacks, attackers can misuse protocols in order to prevent the service from functioning correctly [59]. P2P overlays inherently offer some resilience from certain DoS attacks as they are distributed and thus are mostly free of single points of failure. However, DoS attacks can also be planned to target key peers and nodes to deny a certain file or peer from being distributed in the network. Some bootstrapping mechanisms are easily targeted by DoS attacks, especially in networks using bootstrapping servers and DNS seeds, which account for most P2P overlay networks. DoS attacks can also be made on massive scales via attacker-controlled botnets, giving them the name of Distributed Denial of Service (DDoS) attacks. DDoS prevention approaches are included in this section as DDoS and DoS are often used interchangeably. The rest of this section is dedicated to presenting DoS prevention techniques for bootstrapping mechanisms

Horizontal scaling, when applied to bootstrapping servers and DNS seeds, can effectively enhance resistance to DoS attacks. The rationale behind this approach is straightforward: Greater numbers of available servers translate into a higher number of servers that would need to be targeted to successfully execute a service denial. In effect, horizontal scaling involves supplementing the existing infrastructure with additional servers, aiming to distribute the workload more evenly, improve system redundancy, and bolster overall robustness.

In the study by Loe and Quaglia [6], it was observed that DNS seeds are employed as the primary peer discovery mechanisms in the majority of blockchain networks. Alarmingly, it was discovered that 32% of the networks tested only use one DNS provider for their DNS seeds, making them incredibly susceptible to DNS service denial. In the same study, however, many networks showcased their adoption of horizontal scaling via the availability of bootstrapping peers. Specifically, Bitcoin's DNS seeds resolved 44,077 unique IP addresses. Even though it remains uncertain if all these addresses could be used for bootstrapping, the sheer resources needed to deny service to all these peers would be considerable. Nonetheless, integrating more servers comes with its own challenges, such as increased operating expenses, system complexity, and maintenance demands.

Proof of Work (PoW) puzzles is a highly researched and used set of techniques capable of mitigating the effects of DoS attacks [60]. They represent a specific implementation of resource testing techniques, which are discussed in detail in Section 4.2.3. These techniques necessitate a new participating peer's device to autonomously solve a computationally intensive cryptographic puzzle. The time it takes to solve such puzzles differs substantially. However, it is crucial that the process of verifying the puzzle's solution requires minimal computational effort from the verifier's side. This approach is designed to prevent the potential DoS threat that could be triggered by the influx of numerous puzzle solutions for verification [60].

NKN is an example of a system that employs PoW cryptographic puzzles in the registration process for its nodes. It is an integral component of the ID generation procedure during the bootstrap phase. This action was strategically implemented as a countermeasure to combat a vast amount of spam user creations. These entities were effectively inducing an application-level DoSattack on the registration process for other participants. By incorporating a PoW mechanism, new nodes are obligated to generate hashed IDs that fall below a predetermined threshold, thereby compelling them to cycle through a multitude of hash possibilities until an ID that satisfies the set criteria is discovered. The validation of an ID's legitimacy is a straightforward, non-computational hard task. However, the overall procedure for joining clients may necessitate a time span extending up to several minutes. [61]

Chapter 5 Experiment

This chapter outlines the steps involved in the process of requirement elicitation, design, and implementation of two new bootstrapping mechanisms. Furthermore, measures undertaken to evaluate these mechanisms will be elucidated, thus facilitating a response to the research question posed as RQ2.

5.1 Software

The bootstrapping mechanisms will be constructed on top of Fallang's prototype of the decentralized Tor overlay network [1]. Their prototype was created in Python using a self-altered fork of the Python Tor package, "TorPy." The prototype uses OpenVPN to tunnel and create easy-to-use IP interfaces between peers. Before beginning the development of the bootstrapping mechanisms in this thesis, Fallang's prototype was forked, refactored, and documented for use in the following Gitlab repository:
https://gitlab.com/tor-overlay-network/tor-overlay-base

The software developed during this thesis is open-sourced and available in Gitlab at:
https://gitlab.com/tor-overlay-network/tor-overlay-bootstrapping

It is not intended for production use as it lacks stability. Additionally, there's no assurance that it can effectively safeguard users' integrity and confidentiality. Nevertheless, it serves as a valuable proof of concept and can be utilized by others for future enhancements and advancements.

5.2 Bootstrapping requirements and restrictions

The prerequisites for bootstrapping the Tor overlay network are distinct from the majority of P2P overlay networks. In general, for P2P networks, all you need to know is the identification of a peer node or the method to establish a connection with a bootstrapping node. However, these conditions do not apply to the Tor network in the same way. The overlay uses Tor relays as its substrate and can not connect through users' regular IP addresses. This overlay network requires each pair of users wanting to establish a direct connection to create a relaying connection between them. For such a relaying connection to work, both parties must know about the same relay to connect through and have the same rendezvous cookie. These two pieces of information can be derived from a shared secret between the two peers. The rendezvous cookie can only be used if there is no established connection on the same relay with the same cookie. A connection can be open as long as the relay and the peers are available [1]. The peer acting as the server must connect to the relay first and create a circuit for the connection to be successfully established. In practice, two people wanting to connect must know a shared secret, their peer's current internal IP, and establish a connection within the same 30-second interval. There are multiple dimensions to the bootstrapping of this network. An attempt will be made to improve the bootstrapping mechanism by making it autonomous and eliminating the requirement to know a peer's internal IP address. The mechanism should handle when and where to set up a relay. Efforts will be made to minimize or eliminate the need for pre-known information, making the network easier to use and set up. The network's anonymity characteristic should ideally be preserved by the bootstrapping procedures as well.

The Tor overlay prototype reserves separate tunnels for each pair of directly connected peers, producing a lot of overhead for the host machines as well as the Tor network [1]. This will restrict the number of possible users in the network.

Routing between non-directly connected established peers will not be addressed as this problem is entirely different. Only direct connections between pairs will be created, and they will act as if all peers can reach each other even though they do not have a direct link. This imposes further restrictions like using DHT-based bootstrapping approaches, heavily limiting the pool of available bootstrapping approaches from Chapter 4. However, which peers each new peer will try to create direct links with will need to be defined. This will define the overlay topology of the network. Since the routing capabilities are not present, peers will be chosen as in an unstructured overlay by connecting to peers randomly or to the first available.

The routing between established peers, which enables communication among indirectly connected users, will not be addressed in this thesis, as it entails an entirely different set of problems. The focus will solely be on establishing direct connections between pairs of peers, ignoring connectivity between all peers. This imposes further restrictions like using DHT-based bootstrapping approaches, heavily limiting the pool of available bootstrapping approaches from Chapter 4. However, it is essential to determine the specific peers with which each new peer will attempt to establish direct links, as this will define the overlay topology of the network. Due to the absence of routing capabilities, the overlay will be unstructured, thus either choosing peers by connecting randomly from a list or to the first available peer.

This results in the following requirements for the bootstrapping mechanisms:

1. The bootstrapping mechanism should be autonomous; in other words, when given the set of necessary details and pre-shared information, all users sharing that information should automatically establish a P2P network and sustain connections.
2. The bootstrapping mechanism should accommodate scenarios wherein no other peers exist within the network
3. The amount of pre-shared information needed to join the network should be minimized.
4. The mechanism should have the capacity to facilitate anonymous transports or offer built-in anonymity.
5. Each user should have a unique identifier decoupled from their location and the host device.
6. Every peer within the network should be reachable via a public interface associated with their respective identifier. This requires some sort of relaying of data.
7. Users should possess the means to authenticate their identities while preventing imposters from assuming their personas.
8. Each new peer joining the network should be assigned a unique internal IP address.
9. Ideally, the bootstrapping mechanism should be fully decentralized, obviating the need for users to place trust in any central authority.
10. When joining the network, the new peer will try to connect to any peer in the network.

5.3 The user

As for all software development projects, it is helpful to define the intended usage and users of the software. This will help set the scope and build the best solutions to the problems. The requirements of a bootstrapping mechanism for the Tor overlay network and its restrictions are defined in Section 5.2. Based on these requirements, the users can be defined as a small group of individuals, between 2 and 10, who wish to create an anonymous P2P network, thus providing location and transport confidentiality between them. They should only need to know a shared secret, like a passphrase, to be able to join the overlay, and the bootstrapping mechanism should handle the rest.

5.4 Anonymous peer identifiers

Various approaches will be investigated to facilitate peer discovery while ensuring the authenticity and integrity of users. It is crucial to prevent users from impersonating others. To address this concern, public key signing algorithms will be employed. Each user who wishes to join the network will generate their own Ed25519 signing and verification keys in a random manner. The Ed25519 verification key will serve as their unique identifier within the network. By utilizing randomly and locally generated Ed25519 keys, it is anticipated that no personally identifiable information will be disclosed while still enabling the establishment of authenticity and integrity across multiple sessions and potentially public channels.

5.5 Rendezvous chooser function with asymmetric cryptography

There exist various ways to establish a shared confidential state between two entities. One such approach involves the utilization of asymmetric cryptography, which employs private/public key pairs. This technique lets a pair of peers derive a shared confidential piece of information by employing a customary key agreement algorithm like Diffie-Hellman (DH). The process involves the exchange of public keys between the peers and the utilization of their corresponding private keys in conjunction with the received public keys. It is worth noting that the public keys can be transmitted via any publicly accessible medium, still rendering it computationally unfeasible for unauthorized parties to derive the same shared state. In the experiments, Elliptic Curve X25519 will be employed. It is also known as Curve25519 and provides DH functionality, offering 128 bits of security with a key length of 256 bits (32 bytes). This elliptic curve cryptography scheme affords substantial security despite its relatively compact key length, making it suitable for experimental purposes.

Utilizing the X25519 function and feeding it with the private key from one peer and the public key from the other, the two peers, having already exchanged their public keys can derive a confidentially shared piece of information. Importantly, this function consistently produces the same output when supplied with identical inputs.

The intention is to extend the non-autonomous bootstrapping mechanism proposed and implemented by Fallang, which involved the utilization of a dynamic timestamp changing every 30 seconds, the network's name, and the self-chosen internal IDs of the peers. These components were concatenated into a string such as "2022-03-02T12:00defaultpeer1peer2". Subsequently, this string was piped into a SHA256 hash function for conversion into a hexadecimal representation of that hash. The hexadecimal value was then converted into an integer value to be used as an index for selecting one relay from a predefined list of available relays. The list of relays came from the Tor consensus. To ensure synchronization of the relay options, it is necessary to periodically download the new Tor consensus, which is updated hourly. Every peer in the network needs to have the exact list of relays for a successful connection establishment. This consensus retrieval was performed manually in Fallang's prototype, but an automatic approach was needed for the specific requirements. Fallang termed their rendezvous chooser function the "pairwise algorithm," shown in Figure 5.1.

```
K  = "2022-03-02T12:00defaultpeer1peer2"
hash = sha256(k).hex
index = int(hash) % num_relays
selected_relay = all_relays[index]
```

Figure 5.1: Fallang's "pairwise" rendezvous chooser and bootstrapping mechanism. Shown as pseudo-code

A Tor rendezvous cookie is determined per the hash shown in Figure 5.1. The initial 20 characters of the hash are employed as the rendezvous cookie.

The "pairwise algorithm" proposed by Fallang is altered, aiming to overcome the limitations associated with its fixed 30-second time intervals. Drawing inspiration from the Briar Bramble protocol and leveraging public key cryptography utilizing X25519 keys [49], modifications are introduced that essentially achieve the same objective as the "pairwise algorithm," albeit substituting the concatenated shared string with a shared secret derived from X25519 private/public key pairs. The outcome of the DH key exchange is a shared secret jointly held by the two parties. Subsequently, it undergoes a SHA256 hash operation and is transformed into its hexadecimal representation. The remaining components of the public-key rendezvous-chooser function align with those of the pairwise rendezvous-chooser. While different cryptographic algorithms can be employed to attain specific levels of security, the ones outlined here are deemed suitable for the purposes of the experiments. Figure 5.2 shows the public-key rendezvous-chooser function.

```
My_private_key // 32 byte X25519 value
Peer_public_key  // 32 byte X25519 value

secret = My_private_key.exchange(Peer_public_key)
hash = sha256(secret).hex
index = int(hash) % num_relays
selected_relay = all_relays[index]
```

Figure 5.2: The "public-key" rendezvous chooser function based on asymmetric DH cryptography. Shown as pseudo-code

By implementing this change in the rendezvous function, the requirement for simultaneous initiating circuit establishment by two peers within a 30-second time interval is obviated. Instead, it introduces the challenge of exchanging their respective X25519 public keys. These keys are generated randomly for each connection attempt, serving as ephemeral session keys valid only for a single direct connection between two peers. It should be noted that despite this modification, one of the peers involved in the connection still assumes the role of the server and is responsible for establishing the circuit prior to the client peer's connection to the rendezvous point. This alteration does not eliminate the necessity for proper timing of circuit establishment in the correct order. Additionally, opting to continue using the indexing of relaying approach as the means of deriving a rendezvous allows the network itself to add filtration rules for its list of relays, potentially removing known malicious relays or letting the network choose geographically beneficial relays for a performance gain, albeit in the potential cost of anonymity-level [1].

5.6 Connection handler

Creating an autonomous bootstrapping mechanism requires a dedicated software component to manage the creation, maintenance, and destruction of direct relay connections between peers. This software component will be called a *connection handler*. When provided with the proper secrets and information, this connection handler should be able to establish a new direct connection by creating a subprocess for each of them. Additionally, it should automate the maintenance and monitoring of the number of active connections. It must execute concurrently and not obstruct the operation of other software components. The connection handler will help each peer combat stale connections and churn. Moreover, a connection process should be removed and terminated in the following cases;

1. A failed attempt to connect results in a timeout.

2. An established link is no longer active.

3. There is a connection failure.

4. The internal IP or id of a peer changes.

5. User wants to disconnect from the network.

5.7 The Pub/Sub bootstrapper

Publish subscribe systems as bootstrappers

As shown in the overview presented in Chapter 4, the ability to use multicast would ease bootstrapping. Exploring alternatives seems prudent given the lack of support for IP-multicast over the Internet and its absence of transport anonymity. It's feasible to move up an abstraction level and utilize decentralized application-level multicast systems instead. As described in Chapter 2, Pub/Sub systems are end-system multicast alternatives often used in distributed systems and IoT. They can abstract away the underlying network and let devices communicate through subscriptions of topics. Publish/Subscribe is an architecture where one can subscribe to data via topics. When you are subscribed to a topic, you should get all the data sent to that topic without having to be connected to the publisher. This aligns well with the anonymity requirement for the bootstrapping mechanisms, as Pub/Sub systems decouple the sender and receiver of a message from another, making the transport of the messages an implementation detail.

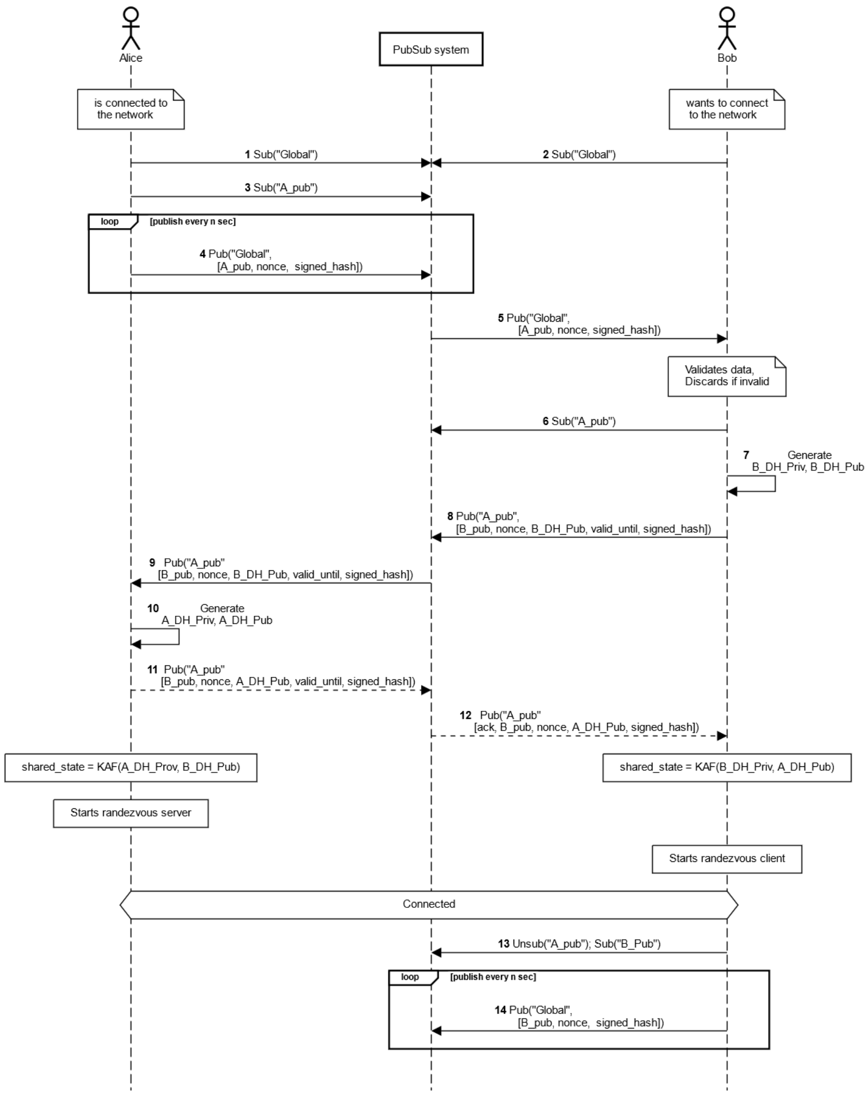
Alice
PubSub system
Bob
is connected to the network
wants to connect to the network
1 Sub("Global")
2 Sub("Global")
3 Sub("A_pub")
loop [publish every n sec]
4 Pub("Global", [A_pub, nonce, signed_hash])
5 Pub("Global", [A_pub, nonce, signed_hash])
Validates data, Discards if invalid
6 Sub("A_pub")
7 Generate B_DH_Priv, B_DH_Pub
8 Pub("A_pub", [B_pub, nonce, B_DH_Pub, valid_until, signed_hash])
9 Pub("A_pub" [B_pub, nonce, B_DH_Pub, valid_until, signed_hash])
10 Generate A_DH_Priv, A_DH_Pub
11 Pub("A_pub" [B_pub, nonce, A_DH_Pub, valid_until, signed_hash])
12 Pub("A_pub" [ack, B_pub, nonce, A_DH_Pub, signed_hash])
shared_state = KAF(A_DH_Prov, B_DH_Pub)
shared_state = KAF(B_DH_Priv, A_DH_Pub)
Starts randezvous server
Starts randezvous client
Connected
13 Unsub("A_pub"); Sub("B_Pub")
loop [publish every n sec]
14 Pub("Global", [B_pub, nonce, signed_hash])

Figure 5.3: Sequence diagram of how the Pub/Sub bootstrapper should work. "A_<Priv/Pub>" means Alice's private or public key Ed25519. If it contains a DH, it means X25519 keys. KAF means Key Agreement Function. The "Pub" and "Sub" functions are publish and subscribe commands.

In Figure 5.3, the utilization of a general Pub/Sub system for bootstrapping in the Tor overlay network is shown. The underlying Pub/Sub system is abstracted away, and only verify-keys are relied upon to prove who the message's sender is. In the figure, Alice is assumed to be already in the network, and Bob wants to connect to the network. The two users only know about the name of the global topic channel and have no other information about the network when they want to join. Alice may have been the first user in the network and thus might have waited for a timeout without no messages in the global topic channel before declaring themselves bootstrapped. When a user is bootstrapped, they should be subscribed to the global channel to create new connections automatically. They should also be subscribed to their own private channel, identified by the user's personal Ed25519 public key, to listen and respond to new users' connection requests. The subscriptions are shown in steps 1 and 3 in the figure. These private channels would let anyone read and publish to them but would function as an interface for communicating with a specific user. Each message is hashed and signed to ensure that a particular user is the message's sender. In step 4, the bootstrapped peer, Alice, is illustrated by periodically publishing their public key, along with a nonce and a signed hash in the global channel. Since Bob is also subscribed to the global channel, he will receive this message, validate it and subscribe to Alice's private channel, as depicted in steps 5-6. Steps 7-9 show that for Bob to create a connection, he will generate DH keys and publish them to Alice's private channel. The Pub/Sub-system will transport the message for him. In step 10-12, Alice gets the message, validates it, generates her own DH keys, and publishes it to her private channel, but also includes Bob's public key. Alice and Bob thus have their respective peer's public DH keys and can use the public-key rendezvous function to derive the intended rendezvous point to connect through. When they have a shared state, they can initiate the connection. Bob will be bootstrapped after a successful connection establishment and can unsubscribe to Alice's private channel and subscribe to his own, as depicted in steps 13-14.

Decentralized IPFS publish-subscribe bootstrapper

Experimenting with a decentralized Pub/Sub system as the base transport for a bootstrapping mechanism involved considering the choice between IPFS Pub/Sub and NKN Pub/Sub. Although NKN's alternative offer a more mature and reliable architecture, IPFS Pub/Sub was chosen because it is fully decentralized and could, in theory, be run over Tor, thus providing anonymity [62]. This feature is not present in NKN. In Figures 5.4, 5.5, 5.6 and 5.7, six state machine diagrams is depicting how this bootstrapper, built on IPFS Pub/Sub, would work. Figure 5.4 shows the main bootstrapper thread, which is the machine that starts the other state machines. It only has three states and controls whether it is bootstrapped into the network. When the state machine starts, it will get or create Ed25519 keys, which will be used for authentication and integrity for all messages sent. Then it will start the global listener state machine, which will be on for the entire session.

The global listener depicted in Figure 5.5 listens to the shared global channel for periodically sent messages by peers in the network. When a message is received, the

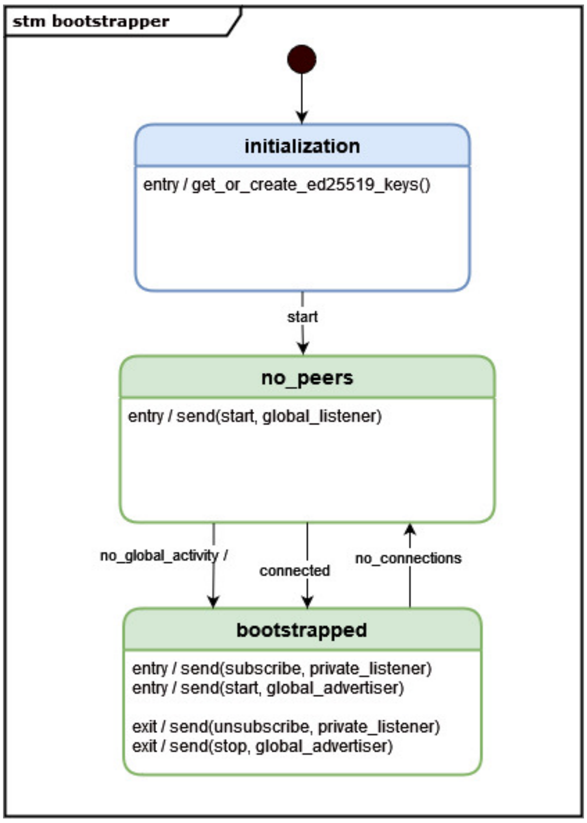
stm bootstrapper
initialization
entry / get_or_create_ed25519_keys()
start
no_peers
entry / send(start, global_listener)
no_global_activity /
connected
no_connections
bootstrapped
entry / send(subscribe, private_listener)
entry / send(start, global_advertiser)
exit / send(unsubscribe, private_listener)
exit / send(stop, global_advertiser)

Figure 5.4: Pub/Sub Bootstrapper state machine. The send functions send events to the state machines in Figure 5.5 and 5.7

corresponding public verify key should be added to a list in a first-in-first-out fashion, which the connecter depicted in Figure 5.6 periodically checks. If the connection handler is configured to want more connections than it currently has, the connecter will start the peer channel state machine shown in the exact figure. This peer channel state machine will subscribe to the peer's channel and publish a connection request unless the channel is too busy. When the peer channel machine gets a response from the peer, it will attempt to open a direct connection to that peer. If the state machine never gets an answer, it will timeout, unsubscribe to the channel topic and terminate itself. There could be multiple peer channel state machines running in tandem.

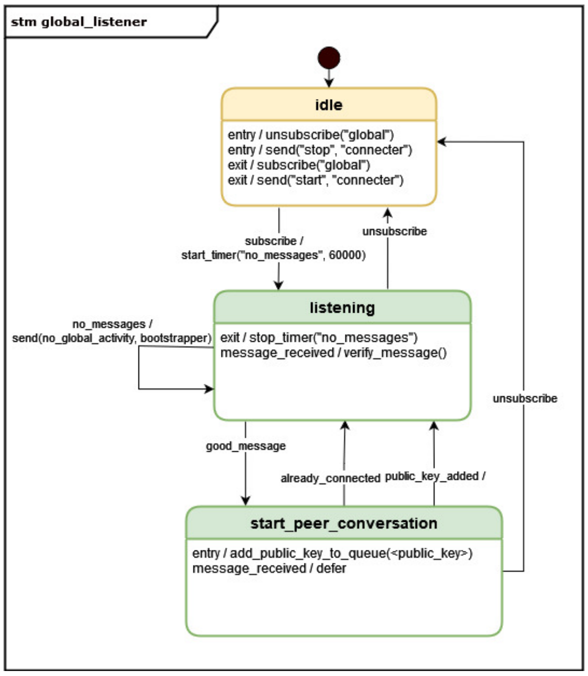
stm global_listener
idle
entry / unsubscribe("global")
entry / send("stop", "connecter")
exit / subscribe("global")
exit / send("start", "connecter")
subscribe /
start_timer("no_messages", 60000)
unsubscribe
listening
no_messages /
send(no_global_activity, bootstrapper)
exit / stop_timer("no_messages")
message_received / verify_message()
unsubscribe
good_message
already_connected
public_key_added /
start_peer_conversation
entry / add_public_key_to_queue(<public_key>)
message_received / defer

Figure 5.5: Pub/Sub Global listener state machine. It is listening for public keys sent in the global channel. The send functions send events to the state machines in Figures 5.6 and 5.4

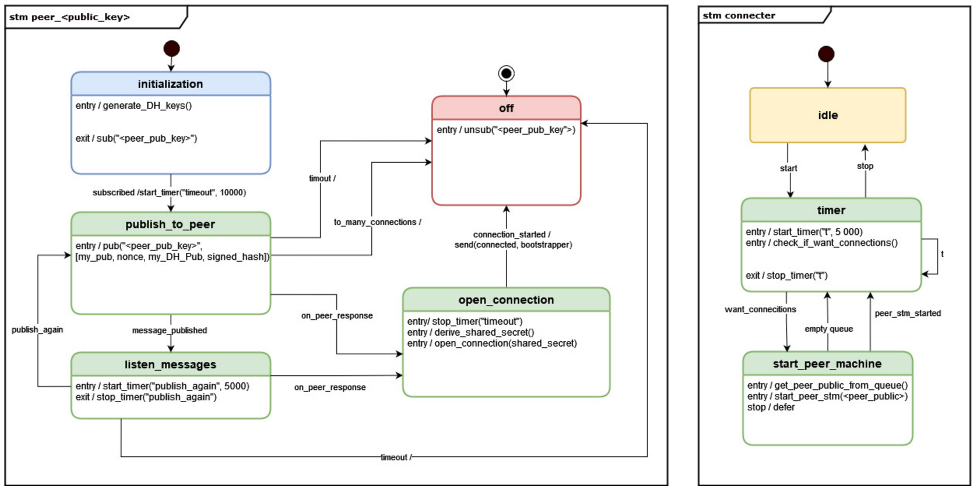
stm peer_<public_key>
initialization
entry / generate_DH_keys()
exit / sub("<peer_pub_key>")
subscribed /start_timer("timeout", 10000)
publish_to_peer
entry / pub("<peer_pub_key>",
[my_pub, nonce, my_DH_Pub, signed_hash])
timout /
to_many_connections /
off
entry / unsub("<peer_pub_key">)
connection_started /
send(connected, bootstrapper)
publish_again
message_published
on_peer_response
open_connection
entry/ stop_timer("timeout")
entry / derive_shared_secret()
entry / open_connection(shared_secret)
listen_messages
entry / start_timer("publish_again", 5000)
exit / stop_timer("publish_again")
on_peer_response
timeout /
stm connecter
idle
start
stop
timer
entry / start_timer("t", 5 000)
entry / check_if_want_connections()
exit / stop_timer("t")
t
want_connecitions
empty queue
peer_stm_started
start_peer_machine
entry / get_peer_public_from_queue()
entry / start_peer_stm(<peer_public>)
stop / defer

Figure 5.6: Peer channel [left] and connecter [right] state machines. The connecter is in control of starting Peer channel state-machines with the "start_peer_stm" function.

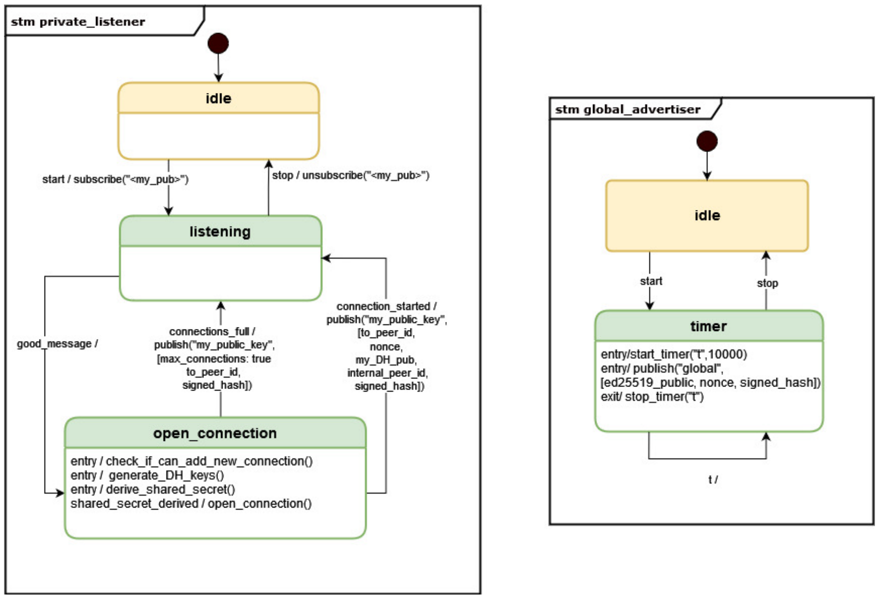
stm private_listener
idle
start / subscribe("<my_pub>")
stop / unsubscribe("<my_pub>")
listening
good_message /
connections_full / publish("my_public_key", [max_connections: true to_peer_id, signed_hash])
connection_started / publish("my_public_key", [to_peer_id, nonce, my_DH_pub, internal_peer_id, signed_hash])
open_connection
entry / check_if_can_add_new_connection()
entry / generate_DH_keys()
entry / derive_shared_secret()
shared_secret_derived / open_connection()
stm global_advertiser
idle
start
stop
timer
entry/start_timer("t",10000)
entry/ publish("global", [ed25519_public, nonce, signed_hash])
exit/ stop_timer("t")
t /

Figure 5.7: Private listener state machine [left] and the Global advertiser state machine [right]. Both are controlled by the Bootstrapper state machine depicted in Figure 5.3

When the first direct connection to another peer is established, or no message is published on the global topic within some pre-defined time frame, the bootstrapper machine should go to the bootstrapped state. It will then start two other state machines depicted in Figure 5.7. The first is the private listener. It subscribes to the user's private channel and responds to connection requests sent by other peer's peer channel state machines. The second is the global advertiser, which periodically publishes the user's verify key in the global channel, letting others know that they are in the network.

Each peer needs unique internal IPs in the network to establish a connection between themselves. In this system, where there is no authority to regulate IP address allocation, a deterministic mapping of users' verify keys to internal IPs can be achieved by converting the verify key to an integer and performing a modulo operation with the number of possible addresses. Fallang's prototype uses OpenVPN and requires each user to set their own and their peer's IP address on connection establishment and allocate a port for each OpenVPN process. To reduce the likelihood that a port is already used, the ports allocated for Dynamic and/or Private Ports are in the range 49152-65535 can be used. This will give around 16383 free ports, thus 16383 possible addresses. The chance of any two users with different verify keys getting the same IP is

$$1 - \frac{(16383-0)}{16383} \cdot \frac{(16383-1)}{16383} \approx 0.0061\% \approx \frac{1}{16393}$$

Since the userbase is defined to be a small group of users of up to ten users, the following probability that at least two users get the same IP address for a network size of 5 can be calculated:

$$1 - \frac{(16383-0)}{16383} \cdot \frac{(16383-1)}{16383} \cdot ... \cdot \frac{(16383-4)}{16383} \approx 0.061\% \approx \frac{1}{1639}$$

and for 10 users:

$$1 - \frac{(16383-0)}{16383} \cdot \frac{(16383-1)}{16383} \cdot ... \cdot \frac{(16383-9)}{16383} \approx 0.26\% \approx \frac{1}{385}$$

Although the probabilities of IP collisions in the scheme are not ideal, they are suitable for the purposes of small experiments. Further enhancements to this approach and different versions are discussed in detail in Section 7.3.

5.8 Onion-type bootstrapper

An additional autonomous bootstrapping mechanism is needed to be able to compare with the Pub/Sub bootstrapper. Since the network is already dependent on Tor, Tor's Onion services could be utilized to make a publicly accessible interface for each peer in the network. This can make it possible to bootstrap anonymously by only knowing a peer's onion address, allowing the network to bootstrap by peers. Onion services could alternatively be used to host bootstrapping servers to make them anonymous.

As mentioned in Chapter 2, a Tor V3 onion services address is a Base32 encoded representation of an Ed25519 public key. This implies that it is possible to programmatically create an onion service for any Ed25519 private-and-public key pairs to which we have access. Furthermore, shared onion services can be designed, allowing anyone in a group with access to the same pre-shared secret to initialize and control them. This could be done by hashing a passphrase with SHA256 and using the hash as the Ed25519 signing key. The Ed25519 verify key can then be derived from the signing key, and an associated onion address would be derivable to all users with the passphrase. The security implications of this are discussed in Section 7.4.3.

With the principles discussed above in mind, an autonomous bootstrapper will be created and named; the "onion-type bootstrapper." In Figure 5.8, the sequence diagram demonstrated how the desired functionality of the onion-type bootstrapper will work. In this bootstrapper, each peer in the network will have their own separate private onion service, which they fully control. Since each user creates their own Ed25519 keys, no one else should be able to masquerade as them without knowing their private Ed25519 signing key. If users know about their peers' verify keys, they also know their associated onion addresses. Knowing the onion address of another peer, they can establish connections using the private onion services as publicly available interfaces, even behind NAT.

The establishment of the network is initiated by the first peer, who is then responsible for the configuration of the collective "Group onion" service. This Group onion will function as the initial point of contact for new users and function much like a DHCP server. Subsequent peers utilize the Group onion to register their public key and retrieve an internal IP in the network. Following the registration, peers may periodically query the Group onion for a map of peers and their IPs. In the event that the user in charge of the Group Onion becomes unavailable or offline, another peer will observe that it is no longer responding and can assume control. The peer taking over would add its latest cache of registered users as a foundation for subsequent registrations. The Group onion could alternatively be implemented as a static, central bootstrapping server to expedite the initial peer's bootstrapping process and minimize potential disruptions induced by their departure. This is discussed in Chapter 7.3.2.

In Figure 5.8, steps 1 to 5, Alice is the first peer in the network and initiates the Group onion service. The Group onion is depicted in Figure 5.8 as the "Group onion" and would exist in addition to each peer's private onion services, shown in the same Figure as "Alice onion" and "Bob onion." As mentioned, the Group onion controls registrations of new peer public key pairs. It maps them to internal IP addresses in a manner similar to how DHCP manages and assigns IP addresses to MAC addresses. As a result, there are no IP collisions and only one user per internal IP address. One may query it for a list of all registered peers, and it will respond with the mapping between public keys and internal IPs for each registered peer. This is shown in steps 16-17 in the figure. Although it is not illustrated, users must renew their registration within a specified time window to remove stale registrations from the network.

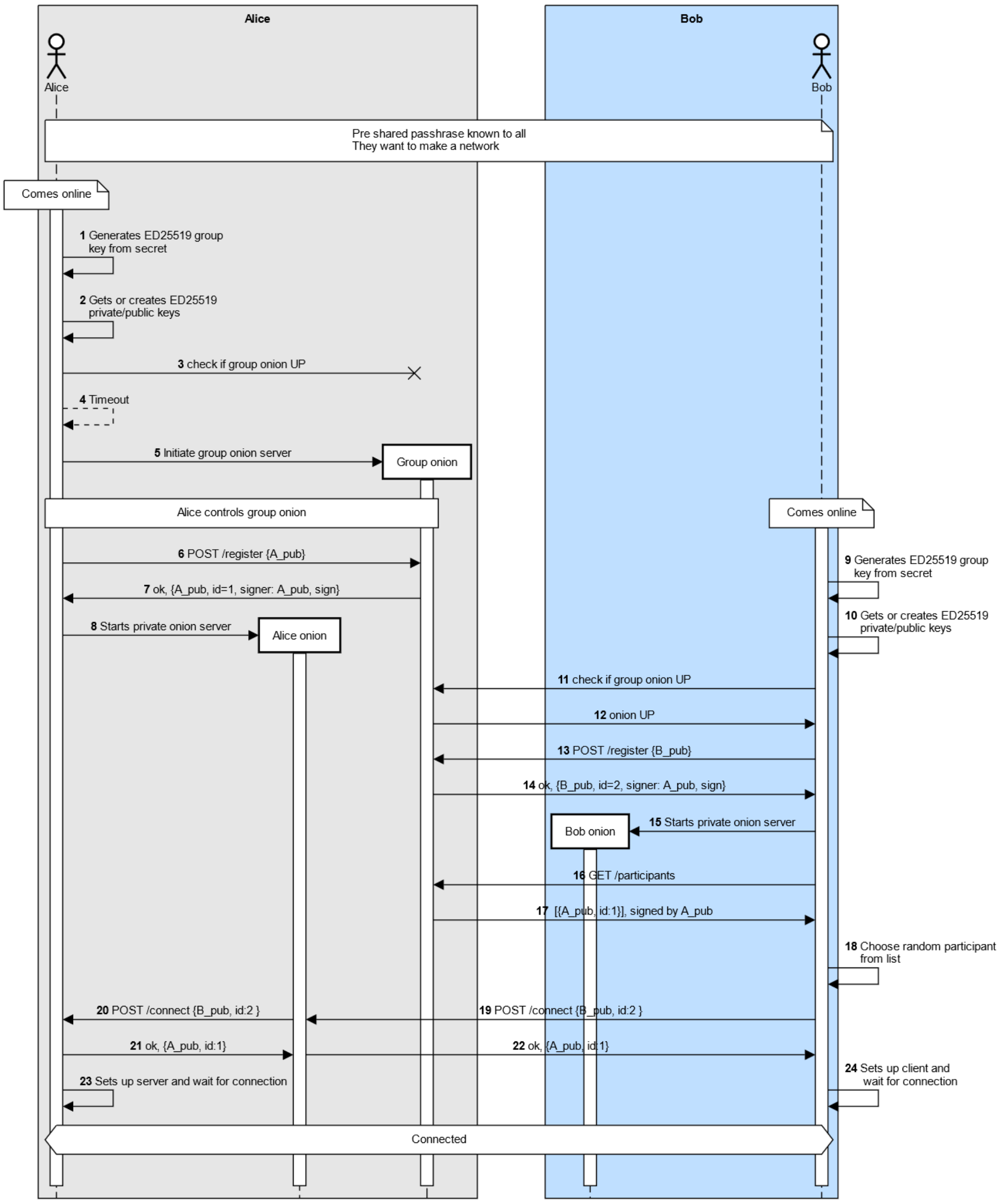
Alice
Bob
Alice
Bob
Pre shared passhrase known to all
They want to make a network
Comes online
1 Generates ED25519 group key from secret
2 Gets or creates ED25519 private/public keys
3 check if group onion UP
4 Timeout
5 Initiate group onion server
Group onion
Alice controls group onion
Comes online
6 POST /register {A_pub}
7 ok, {A_pub, id=1, signer: A_pub, sign}
8 Starts private onion server
Alice onion
9 Generates ED25519 group key from secret
10 Gets or creates ED25519 private/public keys
11 check if group onion UP
12 onion UP
13 POST /register {B_pub}
14 ok, {B_pub, id=2, signer: A_pub, sign}
15 Starts private onion server
Bob onion
16 GET /participants
17 [{A_pub, id:1}], signed by A_pub
18 Choose random participant from list
20 POST /connect {B_pub, id:2 }
19 POST /connect {B_pub, id:2 }
21 ok, {A_pub, id:1}
22 ok, {A_pub, id:1}
23 Sets up server and wait for connection
24 Sets up client and wait for connection
Connected

Figure 5.8: Sequence diagram of how the onion-type bootstrapper should work for the two first peers in the networks. Alice is the first to join the network and gets additional capabilities as the owner of the group onion. Bob uses the group onion to bootstrap himself into the network. Bob's actions also represent all subsequent peers in the network.

Following is a set of state machine diagrams. Figure 5.9 and 5.10 illustrate how such a bootstrapping mechanism could be implemented using state machines. In the first diagram, Figure 5.9, The main bootstrapping logic is depicted into five distinct states with defined transition events between them. The logic for the private and group onions is not depicted since they should be implemented as standard HTTP web servers using Tor Onion services.

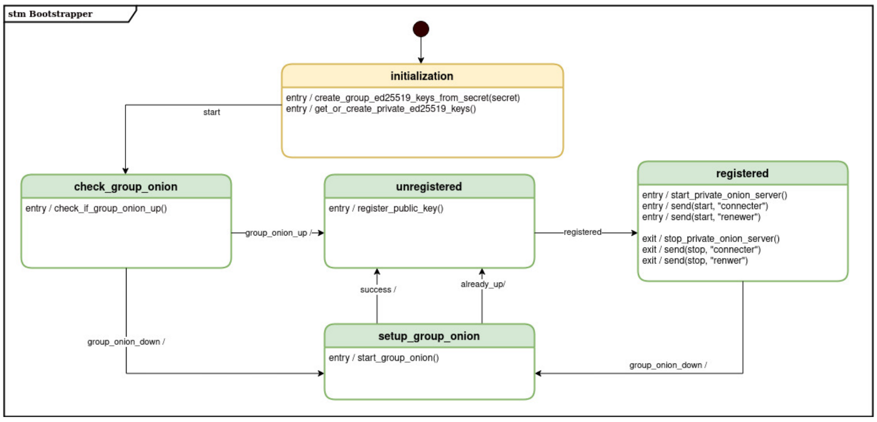
stm Bootstrapper
initialization
entry / create_group_ed25519_keys_from_secret(secret)
entry / get_or_create_private_ed25519_keys()
start
check_group_onion
entry / check_if_group_onion_up()
group_onion_up /
unregistered
entry / register_public_key()
registered
registered
entry / start_private_onion_server()
entry / send(start, "connecter")
entry / send(start, "renewer")
exit / stop_private_onion_server()
exit / send(stop, "connecter")
exit / send(stop, "renwer")
success /
already_up/
group_onion_down /
setup_group_onion
entry / start_group_onion()
group_onion_down /

Figure 5.9: Bootstrapper state machine for the onion bootstrapper. The send functions send events to the state machines in Figure 5.10.

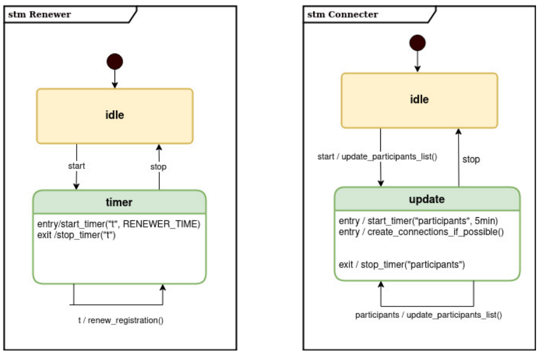
stm Renewer
idle
start
stop
timer
entry/start_timer("t", RENEWER_TIME)
exit /stop_timer("t")
t / renew_registration()
stm Connecter
idle
start / update_participants_list()
stop
update
entry / start_timer("participants", 5min)
entry / create_connections_if_possible()
exit / stop_timer("participants")
participants / update_participants_list()

Figure 5.10: Onion bootstrapper helper state machines. The Renewer is in control of renewing registration. The Connecter regularly fetches the participants list and will try to connect to some participants if needed.

5.9 Tests

To start answering RQ2, empirical measurements of the mechanisms' bootstrapping speed and the amount of failed connection and bootstrapping attempts are needed.

The speed is easy to measure and will tell a lot about the usability of the mechanisms.

To get some context on the rest of the tests, a baseline of the connection speeds will be undertaken by measuring how long it takes two peers to connect without the software as well as with the public key rendezvous chooser function and the connection handler.

In order to provide context for the remaining tests, a baseline for the connection speeds will be established. This will involve measuring the time it takes for two peers to connect without the autonomous bootstrapping mechanism. The measurements will utilize the public key rendezvous chooser function and the connection handler.

Regarding the speed of bootstrapping, the time it takes for the first peer in the network to be ready to establish new connections will be measured. Additionally, the waiting time for subsequent peers to successfully bootstrap into the network will also be measured. The duration before a user is ready to accept a connection request in the bootstrapping mechanisms will vary depending on whether it is the first peer in the network. This measurement will be conducted for network sizes of 0 and 1.

To obtain the main speed results, the time required for a new peer to connect with one of the peers in established networks of 1, 3, and 5 peers will be measured. Additionally, the number of failed connections and bootstrapping attempts will be recorded.

Chapter 6

Results

This chapter provides an overview of the parameters utilized in our experiments and the performance outcomes obtained from the two bootstrapping mechanisms developed in this thesis.

6.1 Implementation

The connection handler and the two proposed bootstrapping mechanisms introduced in Chapter 5 were successfully created. During the development of the bootstrapping mechanisms, IPFS Pub/Sub was marked as deprecated by the IPFS Kubo team [63]. It did, however, continue to function for our experiments but will be entirely removed in later versions.

To programmatically create and destroy Tor onion services, the official Tor python controller library, "Stem" version 1.8.1 [64], is used. It is mostly unmaintained but offers the most comprehensive set of tools for Onion service establishment, maintenance, and monitoring. It is also compatible with the current version of Tor Onion Services, version 3.

6.2 Test results

The experiments were done on Kali Linux (5.9.0-kali1-amd64) virtual machines through the Oracle VM VirtualBox virtualizer on a single desktop computer. Each virtual machine was given 2048 MB of RAM and 1 CPU thread from an AMD Ryzen 5 5600X 3.70 GHz 6-Core Processor. Each peer in the test was running on separate virtual machines, up to 6 virtual machines in the most comprehensive tests.

All tests were done on the live Tor network, with fresh Tor consensus lists automatically fetched every hour. The number of relays may change every hour, but the number of relays to choose from in our tests was always around 5600 +- 10 after filtering out all the Tor relays without nicknames. Fallangs two-hop architecture is still used for the server peer and one-hop for the client peer. The guard nodes in the two-hop circuits were chosen uniformly at random from the same list of filtered available Tor relays.

Configuration

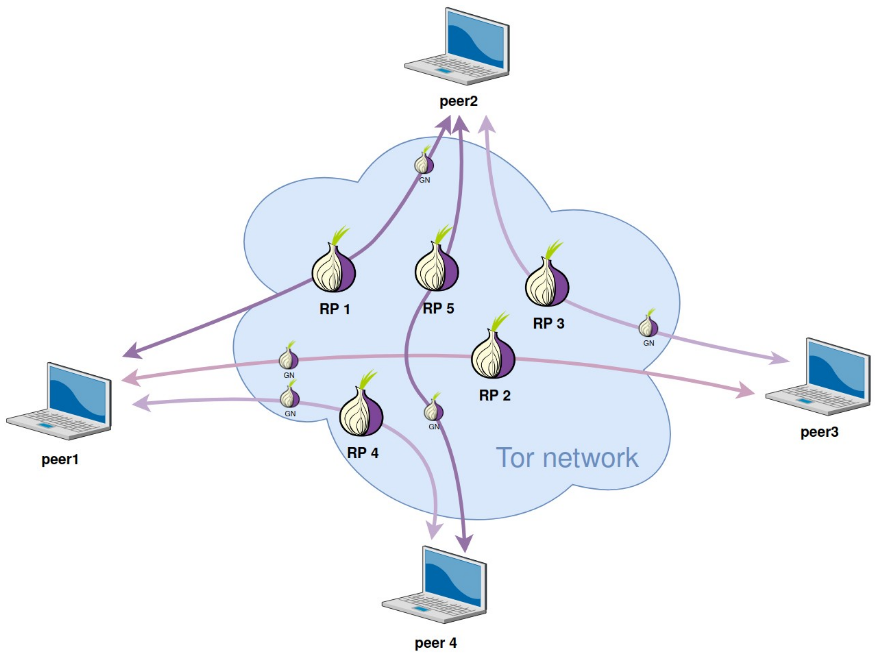

Figure 6.1: One possible outcome of a four-peer network with a minimum of 2 connections per peer. RP is Rendezvous Point. GP is Guard Node. All onions depict a separate Tor relay. RPs are chosen with DH key exchange. GN is chosen uniformly at random

The connection handler for both bootstrapping mechanisms was configured with a preferred and maximum amount of direct connections. In these tests, the preferred amount of direct connections to different peers is 2, and the maximum is 4, which means that the bootstrapper will try to create two direct links whenever possible. It will never attempt to create two direct links to the same peer. After that, it can create two more links if someone needs to connect to them, but it will not actively seek out new connections. This is depicted in Figure 6.1, where four peers have formed a network. In this network, peer4 will not try to connect to peer3 as both have enough connections. This spreads the connection load across all peers and creates a more robust network. If a peer goes offline, it should not take down the whole network.

6.3 Base connection tests

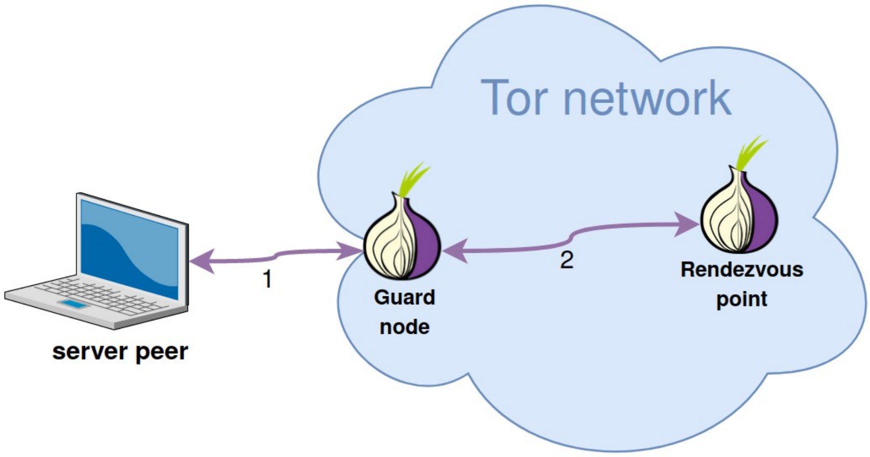

Figure 6.2: Circuit establishment for the server peer. The server peer must first establish a circuit to the Guard node and then to the Rendezvous Point.

Figure 6.2 depicts the steps needed for the results in Table 6.1. Based on the data in Table 6.1, the client wait time will be set to 10 seconds for the cases when the client and server are started simultaneously. This is to set up the connections in the correct order, as the server must connect to the rendezvous before the client and establish a circuit. This should result in a successful connection for most connection attempts.

Table 6.1: Time until the peer acting as a server is ready to create a connection after establishing a circuit via a Tor guard node and a rendezvous point. Guard and rendezvous points were randomly chosen from the list of all available relays.

Average	7.356926489
Min	6.096928
Max	10.649285
Stdev	1.018973182
Percent failures	6.0%

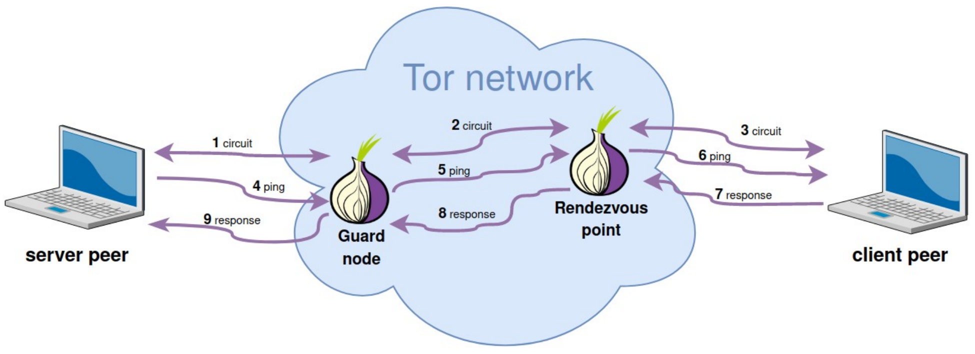

Figure 6.3: Steps needed for connection establishment and first ping between two peers. This does not include the automatic fetching of the Tor consensus list each hour.

Table 6.2 contains the results of how long it took to connect to a peer without the overhead of the bootstrappers. Figure 6.3 shows the steps needed for this. Both the connection handler and the public key rendezvous chooser function were utilized in these tests. The timings are given in seconds, started from the connection handler's creation, and run until the first successful ping between the peers. The server and client were given their respective public keys and started simultaneously. The client had a 10-second wait time meaning that the server had 10 seconds to do steps 1 and 2 before the client peer did step 3 in the figure.

Table 6.2: Time to connect/until the first ping between two peers by only using the public-key rendezvous chooser and the connection handler

Average	26.73255327 s
Min	17.667069 s
Max	44.060099 s
Stdev	5.099299275
Percent failures	8.0%

6.4 Bootstrapping results

The following presents the results gathered from our two bootstrapping mechanisms.

6.4.1 Time before ready to accept connections

The data presented in Tables 6.3 and 6.4 shows the duration required for a new peer to become capable of accepting incoming connection requests. This measures the two bootstrapping mechanisms when the peer intends to join a network with either no pre-existing members or just one member, shown as "0 peers" and "1 peer," respectively. Notably, when the network has no existing peers, the incoming peer becomes the network's initial member and should behave differently than if it were not the case.

In the experiments done on the onion-type bootstrapper, referenced in Table 6.3, distinct network passphrases were employed for every test. This approach facilitated the generation of a new onion service descriptor for each measurement, thereby ensuring that any pre-existing descriptors would not influence the test outcomes.

Table 6.3: Time until a new peer is ready to accept connections request with the Onion-type bootstrapper for a network with no existing peers and a network with one existing peer

	0 peers	1 peer
Avarage	75.441 s	12.353 s
Min	22.184 s	5.0728 s
Max	207.883 s	24.927 s
Stdev	64.867 s	7.109 s

Table 6.4: Time until a peer is ready to accept connections request with the IPFS Pub/Sub bootstrapper for a network with no existing peers and a network with one existing peer. The timeout for not seeing any public keys in the global channel and declaring themselves as the first peer was set to 3· GLOBAL-INTERVAL = 21s.

	0 peers	1 peer
Avarage	21.926 s	14.062 s
Min	21.847 s	3.157 s
Max	21.985 s	22.123 s
Stdev	0.053 s	8.558 s

6.4.2 Time until first connection

Table 6.5: Time until first successful connection measured from starting the onion-type bootstrapper until first successful ping to another peer in an established network of different sizes. Includes the number of failed connection attempts and the number of measurements that do not result in a successful bootstrapping procedure.

Network size	1 peers	3 peers	5 peers
Avarage	46.309 s	49.919 s	43.361 s
Min	22.0158 s	30.432 s	30.430 s
Max	160.362 s	77.525 s	77.451 s
Stdev	40.564 s	15.415 s	15.108 s
Average failures before connecting	0.2	0.4	0.4
Never connected	0.0%	0.0 %	0.0 %

All bootstrapping attempts with the onion bootstrapper were successful, even though every connection attempt was not.

Table 6.6: Time until first successful connection measured from starting the Pub/Sub bootstrapper until first successful ping to another peer in an established network of different sizes. Includes the number of failed connection attempts and the number of measurements that do not result in a successful bootstrapping procedure.

Network size	1 peers	3 peers	5 peers
Avarage	90.715 s	34.748 s	N/A
Min	32.173 s	18.346 s	N/A
Max	180.265 s	78.641 s	N/A
Stdev	49.516 s	24.146 s	N/A
Average failures before connecting	1.5	0.5	N/A
Never connected	10.0%	0.0 %	N/A

The Pub/Sub bootstrapper speed results are shown in Table 6.6. Measurements for the network with 5 peers could not be obtained due to the inability to create a stable network using this bootstrapper. In the case of the network with only one existing peer, attempts to bootstrap into the network for all measurements were unsuccessful even after waiting for 5 minutes.

Chapter 7 Discussion

The experiments yielded results for our bootstrapping mechanisms presented in the previous chapter. This chapter will discuss the findings as sufficient statistical data has been collected to make certain generalizations. Furthermore, the mechanisms will be examined in terms of their architecture, security, and level of anonymity.

7.1 General

Before continuing the discussion, some general aspects of the thesis and our implementations must be addressed.

7.1.1 Overview

In the referenced Chapter 4, an exploration of the difficulties inherent in overlay networks was presented, incorporating a collection of techniques and strategies to address these issues. This overview was framed as a response to RQ1. No new approaches were discovered during the course of the thesis that was previously unknown. However, a deeper understanding of the reasons behind existing approaches was gained. The insights obtained from this review aided in designing and implementing the two bootstrapping mechanisms.

As previously acknowledged in Chapter 1.6, the quality of several references enlisted to facilitate our literature review leaves room for improvement. This deficiency is particularly evident in the case of the references about Bitcoin, IPFS, and NKN's implementations, where their documentation is dispersed across diverse platforms such as forum threads, wiki pages and articles, and GitHub repositories. Despite the aforementioned limitations, the confidence in the quality of these sources remains intact due to the consistency observed between the information presented and the actual open-source codebases. Verification of the codebases has been conducted to support this assurance.

7.1.2 Unoptimized software

The Fallang prototype, which is unoptimized and not designed for speed, primarily serves as a proof of concept [1]. Our implementation builds upon this unoptimized prototype and includes our own slow code. Rewriting the software in languages like C++, Rust, or Go-lang would increase the performance of the application client and could reduce the number of resources needed to run it. There are a lot of local code optimizations possible that could make the Python code faster. For example, switch to queues instead of periodically checking for new public keys in Figure 5.10. However, the software relies on external API calls, and the Tor network and circuit establishments would still be our main performance bottleneck.

After implementing the bootstrapping mechanisms, several improvements were considered to the onion-type bootstrapper. Firstly, the first peer's group onion check could be expedited by initially verifying the availability of the group onion's hidden service descriptor before proceeding. If the descriptor is unavailable, the first peer can immediately set up the group onion instead of waiting for a timeout. Secondly, the architecture could be modified to make the group onion a static server, or bootstrapping node, that can be shared amongst different networks. To improve performance and robustness, the Python library Onionbalance [65] could be utilized for load-balancing across multiple duplicate onion services, albeit at the cost of some centralization. Thirdly, Web-sockets could be integrated with the Onion-type bootstrapper, thus eliminating the need for the static 10 seconds waiting time found in Table 6.1, and using the minimum amount of time possible. Finally, instead of sending a new connection request through the private onion services or private Pub/Sub channels, the connection handler for a peer client could attempt to connect multiple times to a server-peer. These alterations were regarded as potential enhancements. However, the development phase had to be concluded due to time constraints, preventing their implementation.

7.1.3 An exploration of techniques

The primary objective of this thesis has been to explore various approaches to bootstrapping a particular P2P network. This research has resulted in a comprehensive exploration of various technologies, although only the Onion-type and Pub/Sub bootstrappers are explicitly mentioned and had practical implementations. In this context, our endeavors concerning the design of bootstrapping mechanisms utilizing blockchains to distribute peer identifiers in a decentralized and pseudo-anonymous approach are worth highlighting. Unfortunately, due to the relatively small scale of our projected user base, these designs did not progress to the implementation stage.

Several alternative bootstrapping mechanisms are envisioned that would utilize publicly accessible blockchains for the registration of users' peer identifiers within their transactions or through smart contract scripts. To minimize the risk of platform deprecation, a certain level of popularity would be needed for a blockchain to be suitable. It turns out that this approach is predominantly suitable for medium to large networks because of the excessive volume of transactions within such systems while not overloading the network with bootstrapping-related transactions. This considerable transaction throughput with long block times causes an increase in the time to search for peer identifiers. Consequently, the bootstrapping duration could be prohibitively lengthy if used for smaller networks. Furthermore, the level of knowledge required for each peer to register their public keys on a blockchain without facilitating easy traceability back to them is markedly high, exemplified through the exchange of real money for cryptocurrency. This led to the choice of proceeding with the two approaches present in Chapter 5 instead, as they were deemed to be more practical alternatives within the intended scale of our network. However, this approach should be further researched for larger networks.

7.2 Experiment

The evaluation of our base connection tests, as described in Section 6.3, was limited to 50 measurements. These were conducted to determine the optimal wait time for clients during the connection process. The sample size was deemed sufficient for the choice of configuration time, as no significant variations were observed. For the remaining tests presented in Section 6.4, the number of measurements had to be limited to ten per test due to the need for concurrent execution on up to six virtual machines and manual execution and readout of the software. Despite the low number of measurements, the findings offer valuable insights, which will be discussed later. However, automating the setup and tear-down of these networks posed technical challenges. The creation of a separate API for coordinating the process of devices may have impacted the test results and added further development overhead. As a result, the decision was made to develop two distinct bootstrapping mechanisms instead of focusing on a single mechanism for in-depth testing. This approach was considered advantageous as it allowed for the exploration of different technologies and ideas.

According to the results presented in Table 6.1, the average time required for a server peer to establish a circuit to which the client peer can connect was approximately 7.35 seconds with a standard deviation of approximately 1.02 seconds. This mostly explored the existing prototype, with our small alteration of the rendezvous chooser function. Based on this finding, it was decided to set the client's wait time to 10 seconds, which is less than the maximum connection time observed but allows for additional time needed for the client to connect to the circuit. To avoid the need for a more complex implementation requiring an open communication channel like WebSockets, the decision to employ a waiting time scheme for the peer connection process was driven by the challenges encountered in establishing consistent functionality with Onion services. Alternatively, an invitation system could have been implemented, but this would have introduced a lot of complexity and extended the development process. This would, however, be needed if the number of hops in the Tor circuit would be altered.

In Table 6.1, a 6% failure rate in circuit establishment attempts was observed. Fallang did not examine this particular finding, but it is believed that the cause of this phenomenon is likely attributable to the arbitrary selection of defective or overloaded relay nodes. The server client is required to establish a circuit between itself, a randomly selected Guard node, and the rendezvous point. Both Tor relays are chosen randomly, albeit one is chosen combined with another peer via a DH exchange. Either of these relays might become unavailable after the latest Tor consensus update, and currently, no fallback mechanism is in place to address this issue.

7.2.1 Time before ready to accept connections

Section 6.4.1 details the performance metrics associated with the duration required for a new user to begin receiving incoming network connection requests in two different scenarios. The first scenario involves the user joining a network with no other peers, and the second situation occurs when a peer is already present in the network. For both bootstrapping mechanisms, this process requires the establishment of a public interface that can be reached using the peer's public key.

Onion-type results

Table 6.3 depicts the findings associated with the onion-type bootstrapper. In the initial scenario, where no pre-existing peers inhabit the network, a new peer must determine the availability of the Group Onion and initiate it if absent. Both scenarios include initializing their private onion service and registering their public key to the Group onion. Although not explicitly mentioned in the results, the observed time consumption associated with setting up the onion services is minimal and consistent; the bulk of the time consumption arises from the confirmation of the onion service's availability, leading to a considerable discrepancy in setup times. The observed variability is largely due to the time it takes to retrieve an onion service's descriptor, establish Tor circuits, and initiate the first connection to the onion service. This process can be quite lengthy, considering the numerous steps involved, each of which can require a substantial amount of time.

A user must first establish a connection with the Tor network and subsequently query the directory servers, the HSDirs. However, a single HSDir lacks a comprehensive view of all accessible onion services, necessitating lookups in their collective DHT of descriptors [14]. Nevertheless, this process can also prove relatively swift, with the minimum time recorded being approximately 22 seconds in the conducted test; this includes the Group Onion's successful response to an HTTP GET request.

The mere existence of an onion service descriptor does not suffice as definitive proof of the availability of the respective onion service. The service could have been operational in the recent past but may not be currently active, necessitating a comprehensive series of steps before confirming its availability. As referenced in Section 7.1.2, employing this initial existence check can expedite the verification process for the first peer, but this is not implemented.

Our implementation includes the utilization of multiple connection threads, which are engaged concurrently to probe the availability of the onion service. The bootstrapper proceeds by either confirming a positive response to the first successful GET request or awaiting the lapse of their respective time-out periods. As the results show, this leads to a substantial extension in the maximum duration, which can extend to 207 seconds. This prolonged duration is attributable to the necessity of waiting for all the connection threads routed via Tor to reach their time-out limit in the absence of a response.

The considerable time duration is only part of the scenarios with no existing network peers. It is observed in the column named "1 peer" that there is a substantial enhancement in speed when an additional peer wants to join the network and successfully locates an available Group onion. Therefore, the process of confirming an onion service's availability tends to be a lot quicker when the outcome is affirmative rather than negative. This latter scenario represents all subsequent peers joining the network. It involves checking for the availability of the group onion, registering themselves to the group onion, and initiating their private onion service.

Pub/sub results

Table 6.4 presents measurements for the condition "0 peers", where the recorded times are and should be nearly identical for each observation. This is given the fact that the first peer will subscribe to the global channel and, given a pre-set timeout interval of 21 seconds, will not detect any messages as there are no other bootstrapped peers in the channel. The necessity for this timeout lies in its preventative measure; without it, incoming peers could establish connections with other incoming peers, potentially forming isolated, smaller networks within the larger pool of peers intending to join the same network. The slight time discrepancies measured can be attributed to variations in the execution times and the somewhat dynamic resources available for each program execution. These variations influence all of our tests, yet their impact remains insubstantial, as evidenced by these results.

The specific timeout duration of 21 seconds is set somewhat arbitrarily, yet a tripling of the public key advertisement interval was selected based on two considerations. Firstly, this choice serves as a method for minimizing the connection time in scenarios where the first two peers join the network nearly simultaneously; hence both would need to endure the entire timeout duration prior to discovering each other. Secondly, it acknowledges the reality that the IPFS Pub/Sub overlay requires some time to discover other peers subscribed to the global channel, during which packet loss may occur. Thus, the time duration set is believed to be a good compromise optimized for these two scenarios. Nevertheless, an investigation of the network deployed in a real-world context would be needed in order to confirm this.

The maximal duration expended for scenario "1 peer", with a pre-existing peer in the network, is depicted in Table 6.4, registering at 22.12 seconds. This duration is expectantly close to the instance with no pre-existing peers, given that the identical procedure has been executed due to the absence of other discoverable peers. The reason is the challenge of identifying other peers has been delegated to the IPFS's Pub/Sub peer-discovery mechanism. This mechanism does not consistently succeed in locating other peers who subscribe to the same topic and, at times, exhibit lengthy discovery. As previously indicated, it remains uncertain how much the deprecation of the Pub/Sub feature contributes to these results. Nonetheless, it is believed that similar results would have been obtained even without the deprecation. This belief is supported by an issue thread, dated before the deprecation, which indicates that the IPFS Pub/Sub peer discovery mechanism has historically been fragile [66]. This is also outlined in the deprecation warning [63]. Unfortunately, this was not discovered while initially investigating the software and might result in a different direction for the implementation.

7.2.2 Time until first connection

In Section 6.4.2, we presented the duration from the initial execution of a bootstrapper by a new user to the successful transmission and reception of the first ping between them and another peer. This data is illustrated in Table 6.5 and 6.6. Additionally, the frequency of connection failures and the count of measurements that did not result in the peer successfully being bootstrapped into the network are included.

Onion-type results

Table 6.5 demonstrates a consistent average connection time for the onion-type bootstrapper across various network sizes. This consistency holds particularly true for the largest two networks examined, those of sizes 3 and 5, yielding almost indistinguishable results on all statistics presented. However, a divergent pattern emerges in the network of size 1. In this instance, the maximum connection duration was observed to be more than twice that of the others, which is attributable to the number of possible parallel connections pre-set in our connection handler. Given that the smallest network has only one other peer to connect with, the joining peer lacks the opportunity to speed up the bootstrapping process through simultaneous connection requests to connect with multiple peers.

The reason behind the observation that the minimum connection time in the network of size one is lesser than that of sizes 3 and 5 could be explained by two potential factors. One possible explanation could be statistical randomness, which might have been overlooked in our measurements due to the limited sample size. The more plausible explanation is related to the behavior of the initial peer in the network, who will attempt to connect to the joining peer immediately after registration, driven by their own desire to establish connections with others. This behavioral pattern is not replicated in the larger networks, as the initial peer has already reached the capacity of their preferred connection pool.

The aforementioned reasoning may also explain why the average count of failed attempts before successful connections is lower for a network of size 1 compared to networks of sizes 3 and 5. More specifically, increased connection attempts naturally imply a higher probability of connection failures. Thus, the more limited opportunities for connection attempts within the smaller network inherently result in fewer overall connection failures.

Pub/sub results

The results presented in Table 6.5 are not as favorable as anticipated. The networks created by this bootstrapper are unstable; the lack of results for the network of size 5 confirms this. Even with more than double the amount of network establishment attempts than that of the onion bootstrapper, consistent and stable connections could not be achieved for this network size. This is believed to be primarily due to a flawed state machine design and implementation, which could not be resolved within the time constraints of this thesis. It is not attributed to the poor peer-discovery mechanism in IPFS Pub/Sub. This is evident from the consistent connections established in networks with 3 peers. This network size showed a great improvement for all statistics compared to the network of size 1.

In the network of size 1, the faults of the IPFS Pub/Sub peer-discovery mechanism can be observed. During 10% of our conducted tests, the two peers involved failed to receive any published messages about the global topic they were both subscribed to, leading to their inability to establish a connection. In the remaining 90 % of the measurements, a significant number of unsuccessful connection attempts were observed, resulting in high average and maximum values for the bootstrapping speed. Although not explicitly depicted in the results, it was observed that the failed connection attempts were attributed to the server peer's rendezvous circuit not being established prior to the client peer's circuit, indicating that the wait time of 10-seconds determined in Section 6.3 is unsuitable for this particular bootstrapper. The observed phenomenon can be hypothesized to be attributed to the impact of high resource utilization by the IPFS Pub/Sub process on the establishment of Tor circuits. This is because the server peer necessitates the establishment of a circuit to an extra Tor relay compared to the client peer. In situations where the processes responsible for circuit establishment lack the necessary resources, the server circuit process would experience greater time-related delays due to the additional steps it must undertake.

7.2.3 Comparison based on performance metrics

In Table 6.3 and 6.4, although the results for the first peer are substantially better for the Pub/Sub bootstrapper than the onion-type bootstrapper, this is mostly attributed to the architecture and the introduction of the group onion functioning as the entry point and authority in the onion bootstrapper. This makes the onion-type bootstrapper robust against IP collisions, which is a feature the Pub/Sub bootstrapper does not provide.

Moreover, it is worth mentioning that our original idea of utilizing IPFS Pub/Sub in the Pub/Sub bootstrapper was predicated on its operation over the Tor network, which would enhance anonymity. However, implementing this feature proved challenging due to a lack of documentation and the potential need for a central server as the interface between IPFS and the Tor network. If such functionality could be realized, it is anticipated that the Pub/Sub bootstrapper would introduce an increase in all of our speed measurements taken. This would be especially the case for the initial bootstrapping phase of our network, as it would include delays from the Tor circuit establishment.

Based on the performance and fault statistics presented in Chapter 6.4, the Onion-type bootstrapper is clearly the greater choice of the two presented prototypes for our small set of intended users. It has a longer initial initialization phase for the first peer joining the network, but the reliability and consistent low bootstrapping speed for subsequent peers it provides far outweigh this downside for one peer.

7.3 Design and architecture

Although two bootstrapping prototypes were created and some performance results were gathered, they are not without their flaws. This section will address the architectural issues encountered during the design and development process and suggest ways to address these flaws in future revisions.

7.3.1 Pub/Sub architecture

IPFS

The Pub/Sub bootstrapper, while providing an alternative approach to bootstrapping, has the downside of introducing additional external services, such as the IPFS network. This goes against Doyle's proposed sixth decentralized bootstrapping requirement as described in Section 2.4. Still, it is believed believe that the choice of utilizing external services is warranted because of the small size of the intended networks. As described in Section 4.1.3, this is an approach used in some P2P networks as a temporary solution to facilitate reliable and decentralized bootstrapping while growing the user base.

In this implementation, reliance is placed on the up-time and performance of the IPFS Pub/Sub system. However, as observed with the deprecation of the Pub/Sub feature as well as the unreliable topic peer-discovery, this approach can be risky. Moreover, the results obtained from this bootstrapping approach indicate that it may not always be able to receive published messages, even when subscribed to the correct topic at the correct time. Given that the IPFS Pub/Sub feature is currently deprecated and will be removed from the IPFS platform, future revisions of this bootstrapping approach require a different Pub/Sub-provider. This was considered while implementing the prototype, abstracting away most of the underlying IPFS Pub/Sub API, which should make the replacement of the Pub/Sub provider a fairly low-hanging fruit. Still, it was not deemed appropriate to implement other Pub/Sub providers during the implementation period of this thesis because of the other problems that will be mentioned in this section and based on the beliefs that the bootstrapper is unreliable because of our implementation. As an experimental thesis, the objective was to test various bootstrapping approaches and identify suitable methods, rather than striving to make a single approach flawless.

IP collisions

IP collisions are a big problem that was not properly addressed in the Pub/Sub implementation. It arises because the network is built on top of a network, the Tor network, that does not provide an addressing scheme. The lack of a satisfactory solution to this problem further reinforced our decision not to switch out the Pub/Sub provider. Instead, the focus was primarily on the onion-type bootstrapper. Section 5.7 shows that a network with 10 users has approximately a 1/385 chance of at least one IP collision. Although no collisions occurred in any of our measurements, this is an unacceptable height chance if it should have any real-world applications, especially since there is no well-defined strategy for dealing with these collisions. The probability could be significantly improved by allowing more ports and changing how ports are allocated. However, this does not fix the problem; it only moves it to a greater scale. The more users that are in the network, the greater the chance of an IP collision.

Alternatively, a decentralized allocation strategy could be implemented, introduce an authority to the system as was done in the onion bootstrapper, or have a consensus mechanism from the users already in the network.

The issue at hand could be entirely removed by eliminating the use of IP addresses for message routing and applying concepts analogous to IPFS and NKN. Peer identifiers, which have already been developed using Ed25519 public keys, could be used in a similar manner to the addressing schemes utilized by IPFS and NKN. This could be achieved by incorporating a DHT specifically for this purpose. However, the design and implementation of such a routing mechanism fall beyond the scope of this thesis.

Alternatively, potential IP collisions would not be an issue if P2P applications are constructed on the Internet instead of an anonymous overlay like the Tor overlay. Therefore, a similar Pub/Sub bootstrapping strategy could be suitable for small decentralized P2P applications that don't require anonymity, such as multiplayer gaming, collaboration tools, or file-sharing applications. These networks could utilize ISP-allocated IPv6 addresses, eliminating the problem altogether while still enabling direct communications between peers. With IPv6, the need for private channels in our Pub/Sub design would be obsolete, and only the global channel would be necessary to distribute IPv6 addresses.

7.3.2 Onion-type architecture

Group onion

Contrary to the Pub/Sub bootstrapper, the Onion-type bootstrapper bestows a significant advantage of anonymity and removes issues of IP collisions. Despite this advantage, the onion-type bootstrapper introduces the necessary cost of single-peer centralization and vulnerability to a single point of failure. The design of this system is such that a single peer exercises control at any particular time while all peers possess the potential to command the Group onion. Although this approach is not very scalable, it is effective for our intended small-scale networks, offering comprehensive control to the system users.

An alternative approach that was considered involves delegating the responsibilities of the Group onion across a centralized, yet load-balanced, set of onion services. Such services could be accommodated on robust and performant servers. Under this model, peers would not directly control the Group onion; rather, they would interact with a similar API to that of the current system. In the instance a peer decides to join the network using a pre-known passphrase, or network name, they would contact the centralized servers, which are accessible via a static onion address, and supply their network passphrase along with their self-generated Ed25519 public key for each request. It would be possible for different networks to utilize the same set of Group onion servers for bootstrapping by providing different passphrases. This alternate solution could adhere to the current system's register and peer-list retrieval procedures.

This alternative model may be more desirable in situations where there is a lack of trust among network peers, for example, when the network passphrase is shared online on a closed forum. This is because it eliminates the current opportunity for malicious peers to seize control of the group onion by only acquiring the passphrase. However, this model does heighten the risk of existing threats, such as DoS attacks, and introduces new threats, such as the potential for attackers to compromise the onion service and enumerate the list of all established networks. Under this scenario, a DoS attack could impact not just one but all networks using the same Group onion. Consequently, the Group onion's function in this design should be strictly limited to a signaling server, facilitating communication among other entities while minimizing its own operational workload and data storage requirements.

IP storage

The existing method of storing mappings between peers' public keys and their respective internal network IPs does not result in IP collisions. However, it imposes significant computational responsibilities on the peer functioning as the Group onion. Furthermore, a comprehensive strategy for addressing the departure of the group onion-controlling-peer is not yet outlined. Under the present system, the first peer that identifies the Group onion as unresponsive or offline attempts to seize control of it. In case of a successful acquisition, this peer employs its most recent peer cache to avoid complete network disruption. Nevertheless, this rarely culminates in the ideal outcome: a peer gaining control and possessing a cache equivalent to the departed group onion's database. One proposal suggests the inclusion of a mechanism allowing the group onion to notify another peer of its impending departure and coordinate a transfer procedure. This, however, would not rectify instances of unintentional disconnections. Another proposal would be for the Group onion to designate a set of inheritors they would need to communicate with. How to best select these inheritors poses new questions, and newly joining peers could easily side-step them in the queue. The proposed shift from a peer-controlled group onion addresses this concern but also raises additional issues.

A plausible solution could involve altering the internal IPs' storage strategy and employing a DHT that ensures redundancy. In this context, the group onion would retain the IP correlations within the network as opposed to a localized database. This alteration could enhance network robustness and facilitate an improved recovery pathway for instances where the group onion departs from the network. This would permit other peers to assume control of the group onion, thereby reducing the risk of data loss when the previous controller disconnects. Additionally, it would decrease the server load on the group onion, as peers already integrated into the network would not need to directly query the group onion for other network peers. Instead, they could query the DHT, distributing the load across all network peers. However, this design was not implemented due to the absence of a routing strategy among non-directly connected peers. Without such a strategy, the DHT would not scale effectively. Every peer would necessitate a direct link to all other peers, negating some of the advantages provided by a P2P network.

7.4 Security and anonymity

Quantifying the security of our bootstrapping mechanisms is challenging; thus, an examination and discussion of their security features and flaws will be conducted instead.

7.4.1 Rendezvous chooser security

The "public key rendezvous chooser function" has been selected for utilization in the bootstrapping mechanisms, as described in Section 5.5.

While it may be deemed unnecessary for the onion-type bootstrapper, which operates on an end-to-end encrypted channel, the utilization of the public key relay chooser function can help mitigate a specific form of attack. This attack involves one peer attempting to manipulate the other into selecting a rendezvous relay under their control. Therefore, despite the alternative option of having one peer directly send the relay information to the other, it is believed that employing the public key relay chooser function is a sensible measure.

In a two-hop architecture, such an attack could result in the deanonymization of one of the users. By leveraging the public key relay chooser function, it would be significantly more difficult for a malicious peer to influence which rendezvous point is selected, as they would need to generate numerous Curve25519 keys, undergo the exchange process with the peer's public key, and identify a key that computes to their own relay. Given the brief validity period of these DH keys in the connection requests, such an endeavor would be prohibitively challenging.

If the pairwise rendezvous chooser were employed in our Pub/Sub bootstrapper, it would introduce a vulnerability to man-in-the-middle attacks due to the transmission of all data through a public channel. In this scenario, an adversary could exploit the situation by impersonating either of the peers involved or DoS their rendezvous point, utilizing the information readily available from the public channel. However, the threat is effectively mitigated by transitioning to our public-key rendezvous chooser. This is achieved by making it computationally challenging for an attacker to derive the rendezvous point and cookie for the connecting peers when they lack knowledge of their private keys. Furthermore, since all messages are signed with Ed25519, tampering with the DH keys sent is practically infeasible for an attacker without also modifying the public key and signature.

7.4.2 Security of the Pub/Sub bootstrapper

Lack of anonymity

As previously mentioned, our initial concept for this bootstrapper involved the application of IPFS Pub/Sub in our bootstrapper, grounded on the premise of its functional integration over the Tor network, which would add a certain level of anonymity. However, the realization of this feature posed challenges due to the scarcity of documentation and the necessity of a centralized server serving as a conduit between IPFS and the Tor network. If the Tor integration had been implemented, it would remove the existing bootstrapping system's primarily decentralized and distributed character.

Because of this lack of anonymity, the existing architecture of the Pub/Sub bootstrapper has inherent limitations that severely compromise the security and privacy of the entire network. Due to its transparency, the current system configuration effortlessly allows a prospective surveillance entity to obtain an IPFS Pub/Sub peer's IP address. This would provide them with complete visibility into the network interactions of each peer. Consequently, this exposure would undermine the network's foundational principles and diminish its utility. This flaw effectively deems the Pub/Sub bootstrapper ill-equipped as a solution, particularly within the context of the Tor overlay network.

In retrospect, given the decision between employing either NKN or IPFS as our publish/subscribe (Pub/Sub) provider, our preference should have gravitated towards NKN. Although NKN does not provide optimal anonymity, its architecture affords a higher level of anonymity than IPFS. This is attributed to NKN's design, where users depend on their respective relays, which creates difficulty in determining a user's IP address and identity. As elucidated in Chapter 2, NKN provides a time guarantee, on the order of milliseconds, for their Pub/Sub system. An additional consideration is that subscribing to topics in NKN involves a blockchain transaction fee, which may not be desirable for all networks. However, as discussed in 4.2.3, this transaction fee serves as a deterrent against Sybil attacks, a prevalent threat in such networks. Thus, despite the associated costs, this could offer a valuable security advantage for some networks. However, an NKN-based Pub/Sub bootstrapper would not be suitable for deployment in a Tor overlay network due to its strong anonymity requirements. Nonetheless, it could be a viable option to consider for networks that do not have strong anonymity requirements.

Potential attacks

An analysis of the security dimensions of the Pub/Sub bootstrapper, setting aside anonymity concerns, reveals considerable vulnerabilities to DoS and Replay attacks. IPFS Pub/Sub does not incorporate measures to regulate the number of messages published to a topic, presenting a potential threat vector. With a single attack surface, attackers could swarm the network with a flood of messages, leading to the potential destruction of the network. Individual peers may also be susceptible to DoS attacks due to their subscriptions to private channels, which could just as easily be targeted.

Switching to the NKN network would offer certain advantages, principally the distribution of subscribed users into multiple buckets. This system enhances network-wide resilience. However, individual subscribers within the same bucket could still be overwhelmed with messages and be DoS-ed.

The IPFS Pub/Sub system upholds data integrity, employing digital signatures for all messages while assigning unique IDs and timestamps. These measures effectively remove man-in-the-middle attacks. Nevertheless, our bootstrapper implementation houses all its data within the data field of each published message without examining other parts of these messages. This strategy facilitates the interchangeability of the Pub/Sub-provider, which enhances system flexibility. It does, however, come at a cost; although our bootstrapper mitigates man-in-the-middle attacks by signing each message, it lacks the inclusion of timestamps or IDs, rendering the system susceptible to Replay attacks. Attackers could disseminate previously published messages, potentially leading to a lot of unanswered messages and failed connections. Implementing timestamps and restricting the use of a message for connection attempts to only those recently transmitted could effectively neutralize such an attack.

7.4.3 Security of the Onion bootstrapper

Security through obscurity

The onion-type bootstrapper employs the principle of security through obscurity, sacrificing certain aspects of its security in favor of enhanced ease of use. The concept around these networks revolves around the notion that a peer can establish or join a network by solely possessing the passphrase. This approach poses vulnerabilities, as malicious actors can exploit various means to infiltrate the network with the intention of deanonymizing peers, seizing control, or destroying the network. The most obvious method is to acquire the passphrase employed to establish and join the network. When only utilizing a passphrase amongst the users that are going to form an anonymous network, the network is based on a high level of trust amongst those peers. If peers disclose the passphrase to unauthorized individuals or the passphrase is somehow leaked, the network's security drastically decreases. With knowledge of the passphrase, an attacker can seize control over the group onion, thereby exerting effortless dominance over the entire network.

Another method for attackers to join the network is to obtain the group onion address. Currently, there are no preventive measures to hinder the admission of peers who know about the group onion address but lack the passphrase for joining. Consequently, the network becomes susceptible to unauthorized entry if the onion address is leaked or randomly guessed. To address this vulnerability, the implementation of a moving target scheme could be included, where the onion address for the group onion is periodically altered. For instance, one approach is to add the current hour and day in the hashing process of the passphrase. Tor also offers the capability to encrypt the descriptor for an onion service, necessitating the possession of a password or private key to gain access [16]. A separate derived key through a one-way function from the network password could then be used as the descriptor password. Employing these two mitigation strategies concurrently would significantly reduce the likelihood of malicious actors discovering and infiltrating the network.

In the process of deriving the Ed25519 keys for the group onion, the initial step involves hashing the pre-shared passphrase, subsequently utilizing the hash as the private key for the onion service. From this private key, the corresponding public key is derived. This approach is necessitated by the requirement for the private key to adhere to a 256-bit format and to be deterministic for all users within the network. This ensures that any user has the capacity to verify the group onion's availability and assume control if deemed necessary. However, this mechanism presents a potential vulnerability. Should the group onion's address become known, yet the passphrase remains undisclosed, an adversary may speed up the process of deducing the Ed25519 private keys from the public keys. This could be achieved by brute forcing or by correlating pre-computed rainbow tables containing commonly used hashed passphrases and their associated public keys. As a countermeasure to this potential threat, the usage of a passphrase of substantial length is recommended. While the likelihood of a successful attack is extremely low, it may be feasible under certain conditions, especially if attackers are aware that the passphrase's length is only a few characters long. It can also be mitigated by introducing salts to the hashing of the passphrases.

DoS attacks on onion services

It is believed that the principle of security through obscurity plays a substantial role in mitigating the frequency of DoS attacks. The difficulty in executing such an attack is compounded when the attacker is oblivious to the targeted onion service. Additionally, the architecture of Tor is purposefully designed to ensure that the discovery of an onion service is challenging without knowledge of the onion address. Despite the existence of these measures, the Group onion remains vulnerable to DoS attacks in the event of their address being uncovered. This vulnerability stems from the likely limited computing power of the computers hosting it, which would be regular computers rather than powerful servers. To address this vulnerability, potential measurements for DoS mitigation are available in the Tor network, such as introduction point rate-limiting and the ability to define a maximum number of concurrent streams [16]. However, these measures alone may not completely stop a determined attacker from overwhelming the group onion with queries. To further enhance resilience against DoS attacks, implementing rate-limiting at the server level of the group onion can be valuable. This can be achieved by integrating a reverse proxy with built-in rate-limiting capabilities, such as Nginx. By employing such a setup, the group onion can effectively control and limit the rate of incoming requests, mitigating the impact of DoS attacks. Additionally, optimizing the performance of the group onion server client can help maximize the available computing resources. One approach is to rewrite the server using a more efficient programming language, moving away from Python to a language that is known for better performance, such as Rust or C++. This enhancement would help alleviate the strain on Group onion's resources and enhance its overall responsiveness to requests. The same techniques could be utilized to achieve DoS mitigation for private onion services, but this is not deemed as important.

Anonymity provided by the Tor Onion services

By utilizing Tor onion services for our onion-type bootstrapper, the aim is to achieve a high degree of anonymity equivalent to the level offered by Onion services. The Onion Services establish a three-step relay chain to a rendezvous point for both the client connecting and the onion services. This process incorporates more relays compared to the connections between peers, suggesting that it is more difficult to breach user anonymity through the bootstrapping process as compared to direct peer-to-peer connections, which presently only involve two relays. This robust anonymity assurance is based on the premise that no personally identifiable information is shared or unintentionally revealed in the bootstrapping mechanism.

Despite the inability to extract user identity insights from our own server implementations, there is a recognition of the potential for data leakage, particularly given our current use of the Python Flask development server for running onion services. While advantageous for error identification and debugging in developmental contexts, development servers may inadvertently expose system information, creating information-gathering opportunities for potential attackers. Furthermore, since the servers operate on individual peers' personal computers, a successful compromise of the onion services by an unidentified exploit could conceivably result in an attacker assuming control of a peer's computer. Therefore, a comprehensive security review of the server and API should be undertaken if our software is to be further developed or used in a production setting. Running the onion services in a segregated environment, such as a container, is recommended to minimize the impact of a potential breach scenario. This approach confines potential compromise to only the servers, thus safeguarding the rest of the computer and the user's personal files.

While Tor can provide substantial anonymity for its users, it is prone to human error, as users may deliberately or inadvertently disclose their identity or location. Tor's level of provided anonymity is considerable, yet it is not infallible as discussed in Chapter 2; existing methods can be used to deanonymize Tor users, such as traffic correlation analysis. Nevertheless, such attempts necessitate significant effort, particularly for Onion services, and would be simpler to execute on the actual direct peer connections within the Tor overlay network.

One potential measure users could adopt to operate Tor over a Virtual Private Network (VPN), which would substantially complicate traffic correlation at the ISP level. However, this approach would likely lead to an increase in software latency. While this latency increase may not significantly impact the bootstrapping mechanism, it could diminish the efficacy of the applications users seek to operate atop this network.

7.4.4 Group secret sharing

While the Pub/Sub bootstrapper did not incorporate it, both of our initially proposed designs involved the necessity of sharing a group secret, such as a passphrase or network name, prior to establishing connections between users aiming to form an anonymous and concealed network. The specific applications to be executed on this P2P network are yet to be determined, though communication and file sharing are potential functionalities.

In the case of our onion-type bootstrapper, sharing a passphrase among all users seeking to join the network is the only requirement. This approach presents two distinct network types. The first and more probable type pertains to private networks, where the passphrase is exchanged among users who possess mutual knowledge and trust. While this kind of network does not ensure anonymity between its users, it enables the creation of a hidden virtual private P2P network, safeguarding the anonymity of participating peers from external observers and eliminating the need for reliance on centralized services. The passphrase can be shared physically or transmitted through trusted end-to-end encrypted channels to establish a network with the desired individuals.

The second network type is where neither of the users knows the identities of the others. Within this network, the level of trust among peers is significantly low, making the employment of the centralized Group onion server approach, as discussed in Section 7.3.2, more preferable. In this scenario, the passphrase could be publicly disclosed by an individual unconcerned about their own identity or shared within an online forum or anonymous group channel. Such networks could cater to individuals with similar objectives or interests who desire to maintain anonymity throughout their interactions.

7.5 Overall comparison

Based on the analysis of performance and reliability metrics presented earlier in Section 7.2, as well as the comprehensive examination of the design and security features associated with the two bootstrapping mechanisms discussed, our assertion is that the onion-type bootstrapper is the optimal choice among the two alternatives considered. The conventional bootstrapping approaches typically utilized in large-scale peer-to-peer networks are unsuitable for satisfying the unique requirements and limitations of the Tor overlay network. This unsuitability primarily arises from the network's limited scale of users, the essential need for preserving anonymity, and the absence of an appropriate routing protocol.

Chapter 8 Conclusion

This thesis examines existing solutions of bootstrapping approaches while also detailing the principal challenges and potential solutions. We provide a structured overview of these challenges and their solutions, even though they appear to be mostly static since the proliferation of P2P networks in the early years of this millennium. Yet, innovative P2P applications continue to emerge. This overview serves as a guide for new P2P networks electing their bootstrapping mechanisms based on the challenges in their respective networks.

Fallang's anonymous P2P overlay network is built on top of Tor, and despite its inherent constraints, it serves as an effective proof-of-concept. Building on this foundational prototype, we have developed two distinctive autonomous bootstrapping mechanisms, each enabling peers to be contacted externally. The first mechanism was built utilizing the decentralized IPFS Pub/Sub system. A deprecation notice for the IPFS Pub/Sub system came when implementing the bootstrapper, and our attempts to enable this mechanism to operate on an anonymous transport were unsuccessful. The final bootstrapper managed to establish network connections among peers, but the system's reliability fell short of our expectations. This can be attributed to a combination of flawed design and an inappropriate choice of the Pub/Sub system. However, the concept may remain viable in certain network environments where anonymity is not a prerequisite. It must be mentioned that this viability would entail a thorough reevaluation of the Pub/Sub provider. These insights provide an opportunity for future research to improve upon these findings, seeking to uncover new methodologies and designs that can bring us closer to realizing the full potential of such mechanisms.

Our second bootstrapping mechanism offers a reliable and anonymous alternative, capable of significant enhancements and additional functionality, which could make it a fitting choice for this network despite its centralized approach. It employs Tor onion services for each peer and implements security through obscurity, relying on the network passphrase to circumvent notable attacks such as DoS. It is noteworthy that before deploying this mechanism as a credible means of ensuring integrity and confidentiality in a real-world P2P environment, a comprehensive security audit should be conducted.

Chapter 9 Future Work

One of the primary challenges in developing the bootstrapping mechanisms for this thesis was the lack of an appropriate network routing strategy. OpenVPN tunnels were used in the prototype, allowing manual Linux routing via other peers. Yet, a routing strategy based on peer identifiers is crucial for practical usability. We propose using a Distributed Hash Table (DHT) for this, as it offers efficient and scalable routing independent of a physical location with minimal network overhead. This DHT-based approach would address the issue of assigning internal IP addresses in the Pub/Sub bootstrapper and enable Peer-to-peer bootstrapping in the Onion-type bootstrapper, thereby decentralizing the network. A robust open-sourced protocol suite like *libp2p* [24] could be examined for this purpose.

Although it did not work well with our network, we believe that a Pub/Sub solution to bootstrapping could be further explored in other small-scale to medium-scaled networks as a free, decentralized bootstrapping alternative. A stable and decentralized Pub/Sub-provider like NKN [25] could be tested for this purpose.

Transitioning to a system of centralized, load-balanced Group onion servers has been thoroughly discussed. This idea serves as an alternative to the current model for establishing public anonymous networks that operate under conditions of minimal trust between peers. It warrants further exploration to determine the robustness of this approach, particularly in its capacity to support the bootstrapping of multiple parallel and separate networks. Moreover, it could be interesting to explore how decentralized naming services, such as *Namecoin* [67] or *Ethereum Name Service (ENS)* [68], might be leveraged to facilitate the resolution of human-readable Tor .onion domains for these bootstrapping servers. This could potentially function in a manner similar to how DNS seeding is utilized in existing networks.

References

[1] F. Fallang, «Security of dark net overlay networks», M.S. thesis, Department of Information Security, Communication Technology, NTNU – Norwegian University of Science, and Technology, Jun. 2022.

[2] E. T. Midtun, «TTM4502 - Bootstrapping decentralized overlay networks», Department of Information Security, Communication Technology, NTNU – Norwegian University of Science, and Technology, Norway, Trondheim, Tech. Rep., Nov. 2022.

[3] J. P. Vergne, «Decentralized vs. Distributed Organization: Blockchain, Machine Learning and the Future of the Digital Platform», *Organization Theory*, vol. 1, no. 4, pp. 263 178 772 097 705–, 2020, Publisher: SAGE Publications.

[4] «Overlay Networks: An Akamai Perspective», in *Advanced Content Delivery, Streaming, and Cloud Services*, M. Pathan, R. K. Sitaraman, and D. Robinson, Eds., Hoboken, NJ, USA: John Wiley & Sons, Inc., Oct. 2014, pp. 305–328.

[5] W. Galuba and S. Girdzijauskas, «Peer-to-Peer Overlay Networks: Structure, Routing and Maintenance», in *Encyclopedia of Database Systems*, New York, NY: Springer, 2018, pp. 2707–2713.

[6] A. F. Loe and E. A. Quaglia, «You Shall Not Join: A Measurement Study of Cryptocurrency Peer-to-Peer Bootstrapping Techniques», in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London United Kingdom: ACM, Nov. 2019, pp. 2231–2247.

[7] H. Zhang, Y. Wen, *et al.*, *Distributed Hash Table: Theory, Platforms and Applications* (SpringerBriefs in Computer Science). New York, NY: Springer, 2013.

[8] H. Yu, J. Buford, and X. Shen, *Handbook of Peer-to-Peer Networking*, 1. Aufl. New York, NY: Springer-Verlag, 2010, vol. 1.

[9] J. Benet, «IPFS - Content Addressed, Versioned, P2P File System», Jul. 2014.

[10] N. Lab, *NKN: A Scalable Self-Evolving and Self-Incentivized Decentralized Network*, Mar. 2018. [Online]. Available: https://nkn.org/wp-content/uploads/2020/10/NKN_Whitepaper.pdf (last visited: May 10, 2023).

[11] Y. Zhang and L. Grondin, *Tech Design Doc: Distributed Data Transmission Network (DDTN)*, Apr. 2022. [Online]. Available: https://github.com/nknorg/nkn/wiki/Tech-Design-Doc:-Distributed-Data-Transmission-Network-(DDTN) (last visited: May 10, 2023).

[12] R. Dingledine, N. Mathewson, and P. Syverson, «Tor: The Second-Generation Onion Router:» Defense Technical Information Center, Fort Belvoir, VA, Tech. Rep., Jan. 2004.

[13] M. Reed, P. Syverson, and D. Goldschlag, «Anonymous connections and onion routing», *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 482–494, May 1998, Conference Name: IEEE Journal on Selected Areas in Communications.

[14] *Dir-spec.txt - Tor directory protocol, version 3*, en, Jan. 2023. [Online]. Available: https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/dir-spec.txt (last visited: May 8, 2023).

[15] S. Matic, C. Troncoso, and J. Caballero, «Dissecting Tor Bridges: A Security Evaluation of Their Private and Public Infrastructures», in *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2017.

[16] *Rend-spec-v3.txt - torspec - Tor's protocol specifications*, en, Mar. 2023. [Online]. Available: https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/rend-spec-v3.txt (last visited: May 5, 2023).

[17] M. Knoll, A. Wacker, *et al.*, «Decentralized Bootstrapping in Pervasive Applications», in *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*, White Plains, NY, USA: IEEE, Mar. 2007, pp. 589–592.

[18] R. C. Doyle, «Distributed Bootstrapping of Peer-to-Peer Networks», 2008.

[19] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, «Scalable application layer multicast», eng, ser. SIGCOMM '02, Book Title: Applications, Technologies, Architectures, and Protocols for Computer Communication: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications; 19-23 Aug. 2002, ACM, 2002, pp. 205–217.

[20] C. K. Yeo, B.-S. Lee, and M. H. Er, «A survey of application level multicast techniques», *Computer communications*, vol. 27, no. 15, pp. 1547–1568, 2004.

[21] T. Zaarour, A. Bhattacharya, and E. Curry, «OpenPubSub: Supporting Large Semantic Content Spaces in Peer-to-Peer Publish/Subscribe Systems for the Internet of Multimedia Things», *IEEE internet of things journal*, vol. 9, no. 18, pp. 17 640–17 659, Sep. 2022.

[22] S. Profanter, A. Tekat, *et al.*, «OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols», in *2019 IEEE International Conference on Industrial Technology (ICIT)*, ISSN: 2643-2978, Feb. 2019, pp. 955–962.

[23] Protocol Labs, *IPFS/Kubo*, original-date: 2014-06-26T08:14:34Z, May 2023. [Online]. Available: https://github.com/ipfs/kubo (last visited: May 21, 2023).

[24] *Libp2p specification*, original-date: 2016-09-13T15:45:30Z, Jun. 2023. [Online]. Available: https://github.com/libp2p/specs (last visited: Jun. 12, 2023).

[25] Yilun Zhang, *Introducing Decentralized Pub/Sub Based on NKN*, Apr. 2019. [Online]. Available: https://forum.nkn.org/t/introducing-decentralized-pub-sub-based-on-nkn/355 (last visited: Mar. 28, 2023).

[26] A. Zeller, «Academic prototyping (invited tutorial)», in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022, New York, NY, USA: Association for Computing Machinery, Nov. 2022, p. 4.

[27] M. Samadi, «Waterative Model: An Integration of the Waterfall and Iterative Software Development Paradigms», *Database Syst. J*, vol. 10, pp. 75–81, Aug. 2019.

[28] K. A. Gary and M. B. Blake, «C-PLAD-SM: Extending Component Requirements with Use Cases and State Machines», in *Software Engineering Research, Management and Applications*, ser. Studies in Computational Intelligence, R. Lee, Ed., Springer International Publishing, Jun. 2018, pp. 93–106.

[29] M. Knoll, M. Helling, *et al.*, «Bootstrapping Peer-to-Peer Systems Using IRC», in *2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, ISSN: 1524-4547, Jun. 2009, pp. 122–127.

[30] *Modify the bootstrap list | IPFS Docs*, Jul. 2022. [Online]. Available: https://docs.ipfs.tech/how-to/modify-bootstrap-list/ (last visited: May 10, 2023).

[31] *Bittorrent/bootstrap-dht*, original-date: 2013-11-01T22:51:57Z, Aug. 2017. [Online]. Available: https://github.com/bittorrent/bootstrap-dht (last visited: May 10, 2023).

[32] *P2P Network*. [Online]. Available: https://developer.bitcoin.org/devguide/p2p_network.html (last visited: May 12, 2023).

[33] *Ethereum/devp2p - specfcication*, Apr. 2023. [Online]. Available: https://github.com/ethereum/devp2p/blob/master (last visited: May 10, 2023).

[34] G. Dán, N. Carlsson, and I. Chatzidrossos, «Efficient and highly available peer discovery: A case for independent trackers and gossiping», in *2011 IEEE International Conference on Peer-to-Peer Computing*, ISSN: 2161-3567, Aug. 2011, pp. 290–299.

[35] D. Boldt, F. Kaminski, and S. Fischer, «Decentralized Bootstrapping for WebRTC-based P2P Networks», *The Fifth International Conference on Building and Exploring Web Based Environments (WEB2017)*, pp. 17–23, 2017.

[36] R. Matzutt, J. Pennekamp, *et al.*, «Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services», in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, Taipei Taiwan: ACM, Oct. 2020, pp. 531–542.

[37] P. Schutz, S. Gal, *et al.*, «Decentralizing indexing and bootstrapping for online applications», *IET Blockchain*, vol. 1, no. 1, pp. 3–15, 2021.

[38] *Satoshi Client Node Discovery - Bitcoin Wiki*, Dec. 2017. [Online]. Available: https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery#IRC_Addresses (last visited: May 14, 2023).

[39] J. W. Lee, H. Schulzrinne, *et al.*, «Bootstrapping large-scale DHT networks», en, in *Proceedings of the 2007 ACM CoNEXT conference on - CoNEXT '07*, New York, New York: ACM Press, 2007, p. 1. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1364654.1364731 (last visited: Apr. 2, 2023).

[40] C. Cramer, K. Kutzner, and T. Fuhrmann, «Bootstrapping locality-aware P2P networks», in *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004) (IEEE Cat. No.04EX955)*, vol. 1, Nov. 2004, pp. 357–361.

[41] *How IPFS works - IPFS Docs*, May 2023. [Online]. Available: https://docs.ipfs.tech/concepts/how-ipfs-works/ (last visited: May 30, 2023).

[42] M. Ogden, K. McKelvey, and M. B. Madsen, «Dat-distributed dataset synchronization and versioning», *Open Science Framework*, 2017. [Online]. Available: https://github.com/dat-ecosystem-archive/whitepaper/blob/master/dat-paper.pdf.

[43] J. Dinger and O. P. Waldhorst, «Decentralized Bootstrapping of P2P Systems: A Practical View», in *NETWORKING 2009: 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings 8*, Springer Berlin Heidelberg, 2009, pp. 703–715.

[44] D. Stutzbach and R. Rejaie, «Understanding churn in peer-to-peer networks», in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, Association for Computing Machinery, Oct. 2006, pp. 189–202.

[45] D. I. Wolinsky, P. S. Juste, *et al.*, «Addressing the P2P Bootstrap Problem for Small Networks», in *IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, IEEE, Apr. 2010.

[46] R. Lee, S. Kiesler, *et al.*, «Anonymity, privacy, and security online», *Pew Research Center*, vol. 5, 2013.

[47] CryptoRekt, «Official Verge Blackpaper 5.0», no. 5, Jan. 2019. [Online]. Available: https://vergecurrency.com/static/blackpaper/verge-blackpaper-v5.0.pdf.

[48] *Home · Wiki · briar / briar*, en, Feb. 2023. [Online]. Available: https://code.briarproject.org/briar/briar/-/wikis/home (last visited: Jun. 9, 2023).

[49] *Bramble Rendezvous Protocol, version 0 - Briar Project*, Jun. 2022. [Online]. Available: https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BRP.md (last visited: Feb. 16, 2023).

[50] *Speek-App/Speek - protocol specification*, May 2023. [Online]. Available: https://github.com/Speek-App/Speek/blob/main/doc/protocol.md (last visited: May 14, 2023).

[51] L. M. Tanczer, R. J. Deibert, *et al.*, «Online Surveillance, Censorship, and Encryption in Academia», *International studies perspectives*, vol. 21, no. 1, pp. 1–36, 2020, Publisher: Wiley.

[52] I. Clarke, O. Sandberg, *et al.*, «Freenet: A Distributed Anonymous Information Storage and Retrieval System», ser. Lecture Notes in Computer Science, vol. 2009, Berlin, Heidelberg: Springer Berlin Heidelberg, Mar. 2001, pp. 46–66.

[53] Z. Trifa and M. Khemakhem, «Sybil Nodes as a Mitigation Strategy Against Sybil Attack», *Procedia computer science*, vol. 32, pp. 1135–1140, 2014, Publisher: Elsevier B.V.

[54] R. John, J. P. Cherian, and J. J. Kizhakkethottam, «A survey of techniques to prevent sybil attacks», in *2015 International Conference on Soft-Computing and Networks Security (ICSNS)*, IEEE, Feb. 2015, pp. 1–6.

[55] Z. Trifa and M. Khemakhem, «Mitigation of Sybil attacks in Structured P2P Overlay Networks», in *2012 Eighth international conference on semantics, knowledge and grids*, IEEE, Jan. 2012, pp. 245–248.

[56] K. Haribabu, C. Hota, and Saravana, «Detecting Sybils in Peer-to-Peer File Replication Systems», in *Information Security and Digital Forensics: First International Conference, ISDF 2009, London, United Kingdom, September 7-9, 2009, Revised Selected Papers 1*, D. Weerasinghe, Ed., vol. 41, Springer Berlin Heidelberg, 2010, pp. 123–134.

[57] W. Zang, P. Zhang, *et al.*, «Detecting Sybil Nodes in Anonymous Communication Systems», *Procedia Computer Science*, vol. 17, pp. 861–869, 2013, Publisher: Elsevier B.V.

[58] M. Castro, P. Druschel, *et al.*, «Secure routing for structured peer-to-peer overlay networks», *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 299–314, 2002.

[59] B. Awerbuch and C. Scheideler, «A Denial-of-Service Resistant DHT», in *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, ser. Lecture Notes in Computer Science, vol. 4731, Springer Berlin Heidelberg, 2007, pp. 370–371.

[60] H. M. J. Almohri, M. Almutawa, *et al.*, «A Client Bootstrapping Protocol for DoS Attack Mitigation on Entry Point Services in the Cloud», *Security and Communication Networks*, vol. 2020, pp. 1–12, Jul. 2020.

[61] Yilun Zhang, *[NKP-0014] Use PoW to prevent generate ID txn spam*, Aug. 2019. [Online]. Available: https://forum.nkn.org/t/nkp-0014-use-pow-to-prevent-generate-id-txn-spam/1668 (last visited: May 23, 2023).

[62] *Lesson: Run IPFS over Tor transport (experimental)*, Mar. 2020. [Online]. Available: https://dweb-primer.ipfs.io/avenues-for-access/tor-transport (last visited: May 30, 2023).

[63] *Deprecate then Remove /api/v0/pubsub/* RPC API and ‘ipfs pubsub‘ Commands · Issue #9717 · ipfs/kubo*, Mar. 2023. [Online]. Available: https://github.com/ipfs/kubo/issues/9717 (last visited: May 2, 2023).

[64] *Stem 1.8.1 Docs*, Sep. 2022. [Online]. Available: https://stem.torproject.org/ (last visited: May 8, 2023).

[65] *Onionbalance 0.2.1 documentation*. [Online]. Available: https://onionbalance.readthedocs.io/en/latest/ (last visited: May 9, 2023).

[66] *Pubsub does not always find other peers · Issue #3745 · ipfs/kubo*, Mar. 2017. [Online]. Available: https://github.com/ipfs/kubo/issues/3745 (last visited: May 25, 2023).

[67] *Namecoin*. [Online]. Available: https://www.namecoin.org/ (last visited: Mar. 15, 2023).

[68] *Introduction - ENS Documentation*. [Online]. Available: https://docs.ens.domains/ (last visited: Mar. 15, 2023).