Fredrik Fallang

Security of dark net overlay networks

Master's thesis in Information Security
Supervisor: Lasse Øverlier
June 2022

Master's thesis

NTNU
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

Fredrik Fallang

Security of dark net overlay networks

Master's thesis in Information Security
Supervisor: Lasse Øverlier
June 2022

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

Abstract

In this master thesis we present a new way of building decentralized anonymous peer-to-peer networks. Utilizing the Tor network relays in a novel way, we are able to build our network atop of existing infrastructure. The system is tested both in its ability to transmit data between peers and how it can be combined with progressive routing strategies. We isolate our tunnel protocol by testing it locally between Tor relays we control. We also run the test on the live network to verify our initial findings and to have a real world example. Security properties of the system are discussed in light of the networks architecture.

We implement a prototype of our design on Tor rendezvous points mechanisms, allowing for a decentralized bootstrapping functionality. Our design allows for a two hop relay chain between the peers in our network. With further work this could likely be decreased to one. This is the minimum of possible relays with our system, but several relays are recommended when a strong degree of anonymity is desired. The performance tests showed a throughput maximum in our implementation when tested on local relays. However, the results from the real world did not diverge that much from the local test, showing that we are able to utilize the live Tor relays in an efficient way. We show that with a proper implementation, the network design would likely yield consistently better results.

Sammendrag

I denne masteroppgaen blir det presentert en metode for å opprette desentraliserte likemannsnettverk. Ved å benytte Tor rutere på en ny måte vi kan bygge et nettverk ovenpå eksisterende infrastruktur. Systeme blir testet både i sin evne til å sende data, men også hvordan det kan kombineres med progressive rutingstrategier. Vi isolerer tunnelprotokollen ved å teste den lokalt mellom Tor routere som er i våres hende. Testen utføres også på det offentlige Tor nettverket for å verifisere funn og prøve det i den ekte verden. Sikkerhetsegenskapene til systemet blir diskutert i lyset av nettverksarkitekturen.

Systemets design resulterte i en prototype bygget oppå Tor sin rendezvouspunkt mekanisme. Dette gjorde oss i stand til å desentralisert oppkoble systemets likemenn. Designet tillater for to hopp i relékjeden mellom likemenn i nettverket. Fremtidig arbeid kan resultere i støtte for ett hopp. Dette er minimum av hva løsningen støtter, og flere hopp er ofte ønskelig og anbefalt i brukstilfeller hvor anonymitet verdsettes. Ytelsestestene avslørte en tydelig topp i vår implementasjon, når testet lokalt i optimale forhold. Resultatene fra den ekte verden divergerte ikke så mye fra denne toppen, som tyder på at vi evner å utnytte Tor ruterene på en effektiv måte. Dette kan bety at vi ved en bedre implementasjon kan oppnå enda bedre resultater som ikke begrenses av Tor nettverket.

Acknowledgements

I would like to thank my supervisor Lasse Øverlier. Your knowledge of the Tor network made working on this thesis intriguing. It would not have been possible without your precise and motivational guidance.

Thanks to family and friends who helped me both directly and indirectly with this thesis. A special thanks to Tiril and Mathias for the camaraderie throughout these last years of my degree.

And last but not least, a special thanks to my partner in crime Kristin. Thanks for filling my life with laughter, challenges and unforgettable memories.

Oslo, 2022-06-01 Fredrik Fallang

Contents

Acronyms

DHT Distributed hash table. 10

HS Hidden service. 2, 11, 40, 46, 47

HSDir Hidden service directory. 11, 40

I2P Invisible Internet Project. 2

RC Rendezvous cookie. 11

RP Rendezvous point. 11, 12, 18, 20, 23, 29

Chapter 1

Introduction

Inspired by the relaying functionality in the Tor project and decentralized peer-to-peer overlay network concepts, we create a semi-anonymous peer-to-peer network design. Utilizing Tor relays gives us the added benefit of an already deployed anonymity network. To be able to use these in a novel way is crucial for creating an efficient network topology. This is because of the extra connectivity required by a peer-to-peer network.

This thesis should be seen as a further exploration of the work done at Harvard by Geoffry Goddel et al.[1]. They designed and developed the Blossom network, which bears many similarities to our proposed design. They urge to further investigation, specifically in regard to performance and network propagation. After we have implemented our own version of the network, we test some of the metrics they mention.

By creating a peer-to-peer overlay network we allow users to communicate directly with each other, while still being able to achieve anonymity. The benefit of an overlay network is the ability to route traffic in a logical way on top of the underlying network. It is this property we utilize in our system by allowing the users of the network to send their traffic indirectly to the recipient, forwarded through a selected number of relays and overlay routers. This enables applications to craft tunnels to fulfill its own requirements in regard to anonymity.

We investigate traffic analysis attacks and mitigations in regard to our network. Low latency networks have historically been vulnerable to traffic analysis, due to the use of correlation algorithms[2][3][4]. We also implement different methods to combat an adversary that has eavesdropping capability.

Finally we test our implementation with common network performance tests. These tests give us an indication of what such a network can achieve, but also verify that such a network can be realized on top of existing infrastructure.

1.1 Keywords

Peer-to-peer, Tor, onion routing, overlay networks, anonymous, latency, Blossom, decentralized, traffic analysis

1.2 Problem description

There exist few active peer-to-peer networks that provide both a decent amount of anonymity and low latency. In academic papers, we find a plethora of proposed solutions[5] [6] [7]. Although these generally provide good guarantees of anonymity when they reach a critical mass of users, few or none have reached this point, making deanonymization a real possibility. Tor, on the other hand has massive adoption, but is not strictly speaking a peer-to-peer network as seen from the users' perspective. Invisible Internet Project (I2P) also has a large amount of routers, but has undesirably high latency.

The Tor network is composed of relays which forward traffic on behalf of its users in an unpredictable way for an external observer. When using the official Tor client, it creates circuits which adheres to the client-server model, and not a peer-to-peer structure. Since each relay only forwards data, it should be possible to instruct the relay to forward traffic in other ways than its default configuration. To be able to utilize this, we need to take a deep dive into how the Tor relays function, with a particular focus on how they can be orchestrated in a way that allows for nontraditional data flow.

Functionality for hosting anonymous services in the Tor network is already implemented in the form of a Hidden service (HS), which can supply the Tor users with functionality to communicate with each other through a central anonymous service. The HS follows the client-server model and as such contains several potential vulnerabilities, some of which have been used to locate and deanonymize the service. This poses a threat to all the users of the service, since a hostile takeover could potentially expose the users.

1.3 Justification, motivation and benefits

The intended users of the presented solutions are individuals that have a need to communicate anonymously in a decentralized fashion. Tor users today are forced to broadly follow the client-server model with other users on the Tor network when wanting to communicate on the dark net. This is because of how hidden services are incorporated in the network. When a hidden service is taken down or unavailable, the entire user group is unable to communicate. With a decentralized topology, or multiple central relaying nodes, the users are able communicate even if some nodes are disconnected. IRC[8] provides an example which consists of a central server relaying the chat to the clients. Should the server become unavailable, all clients would be unable to communicate. If each client was able to directly communicate with each other, they could send messages independently of a central server. In short, our implementation is intended for use cases where data should flow directly between the participants, and anonymity is critical.

Using existing infrastructure to create new and potentially better suited topologies could prove to be an efficient approach compared to deploying a new one. It could be possible to deploy such a network faster, due to the existing infrastructure. In addition, use cases which historically could not efficiently be transferred through

an anonymizing network, could with some tweaking be enabled to. Online video conferences, audio calls and gaming are amongst some time crucial applications that historically would have bad performance in a large relaying setup.

In later years, the need for anti blocking and censorship circumvention has shown itself crucial for the modern web, particularly in certain countries during times of emergency and crisis. When flow of information is needed the most, some ISPs are forced by their governments to choke the connection. Tor is already an existing solution to this problem. However, Tor does not have said native peer-to-peer support.

By implementing a way for users to communicate directly with each other on Tor, we can combine the anonymous properties of Tor with peer-to-peer architecture. We consider multiple ways to create the network in an effort to utilize Tor relays in the most efficient way possible. Creating a protocol for peers to connect will also make it possible to eliminate the need for any central entities that are vulnerable to attacks. For some features, a central directory server is preferable, but by avoiding it we minimize the attack surface of the peer-to-peer network.

Even though anonymous peer-to-peer systems already exist, none of them directly utilize the Tor network[7]. Tor is among the largest overlay networks accessible to any user. This kind of infrastructure is valuable in regards to censorship and correlation attacks. Instead of proposing something new, we can use existing infrastructure to deploy our network, which maximizes the chance it ends up as something that gets implemented. To build such a network from the ground up can be challenging. An overlay network is dependant on routers deployed on geographically diverse locations. To mitigate against surveillance and correlation attacks, these severs can not belong to any single entity. To build a global network that is voluntarily operated is therefore a time consuming and difficult task. Luckily, the Tor project already completed this task.

1.4 Research question

1. How can existing anonymous networks facilitate a peer-to-peer overlay network structure?
2. Is it possible to bootstrap the individual tunnels between peers in a decentralized fashion?
3. What performance can such a network achieve between single peers and in more complex configurations?

1.5 Scope and contributions

This thesis will primarily contribute three things. Firstly, a design for a peer-to-peer network that utilizes Tor relaying nodes. This includes a tunnel design, used between each peer, but also a bootstrapping mechanism that makes the network able to communicate autonomously. The second is the proof-of-concept software we create, which demonstrates that this approach is feasible. Finally we carry out a set of

performance tests on the software we have deployed. These results are only meant as an indication of what such a network can do, both performance and topology wise.

The focus of this thesis is the design and security properties of the proposed network. Trust and distribution of preshared secrets are therefore considered out of scope. However, well tested solutions to these problems exist, as the literature study shows. Advanced automated routing in the overlay network is only briefly discussed, and the thesis uses a simple routing approach. Since we create a standard IP network, regular routing protocols should integrate without massive investment.

1.6 Ethical considerations

Using Tor relays in a different way than intended could prove to be dangerous in regard to anonymity. There exists research regarding what anonymity Tor can provide with the default configuration[9] [10]. Since we use a different setup it is likely that these proved properties no longer hold. The developed system can therefore not be recommended before a thorough investigation is performed, with focus on which properties and configuration give a tolerable amount of anonymity.

1.7 Thesis outline

In chapter 2 we present the background of overlay networks with a special focus on the Tor network. More specifically, we investigate peer-to-peer networks with an emphasis on those that support anonymity-enhancing methods. Tor is presented as a network that can be used to facilitate our implementation. Chapter 3 presents the method by which we realize our implementation and how we execute the performance tests. Chapter 4 presents different designs and possibilities for how to create a peer-to-peer network atop of Tor. We also present different network topologies where we can conduct different tests. In chapter 5, we present the results from the chosen design and performance tests. In chapter 6 we discuss performance results, limitations and possibilities with our prototype. Finally we present a conclusion in chapter 7, followed by suggestions for further research in chapter 8.

Chapter 2

Background

We present here relevant technologies and earlier efforts at similar experiments. Concepts such as overlay networks and peer-to-peer networks create the platform that we intend to further develop. We intend to utilize Tor to realise our experiment, so we present both Tor at a glance, but also the Tor building blocks relevant for our prototype. This is presented in more details in section 2.3.

Central terms in our work are *decentralized* and *anonymity*. For the prior we will use the same definition as presented by Radner [11]. *Decentralized* is presented as a lack of central or coordinating entity, that leads to decision making based on a limited set of available information. It has a special meaning for overlay networks, but this definition works as a general approach. Anonymity is somewhat related to decentralization, since decentralization can offer a higher degree of anonymity. Anonymity can be sub grouped into two categories: data anonymity and connection anonymity[12]. Data anonymity is concerned with the ability to correlate a data set to an originator. Connection anonymity is related to detecting the originator and receiver of a data stream. It is the latter that is the primary focus of this thesis. When mentioning anonymity, we refer to the connection anonymity unless specified otherwise. Another relevant term used is dark net. It is a loosely defined popular term for everything that is not directly accessible on the internet[13]. To be able to reach it, you need specially crafted software.

2.1 Overlay networks

An overlay network is, as the name implies, a network built on top of another existing network[14]. The supporting network underneath, often referred to as the substrate network, is the foundation for the overlay. The substrate network may perhaps route and shape traffic in an unwanted way. By placing routers in central places in the substrate, we gain the ability to control where traffic flows and distributes. There are not many requirements for the substrate network, other than it must be able to transfer data between the logical routers of the overlay network. The overlay routers should be placed in a logical way, to best suit the users and the purpose of the overlay network. The raison d'être for overlay networks is their ability to route traffic in an

abstract way, independent of the network it is built upon. Another reason is the ability to build an abstraction layer on the existing network. An example of this is how the packet switched internet was delivered over existing line switched telephone lines.

Overcast is among the first described overlay networks that uses internet as a substrate. The overcast network was built to achieve multicast across the internet in an efficient way[14]. By placing nodes at practical locations in the world, they were able to stream data to large geographical areas without using multiple streams from a central server. Instead they relayed traffic to overlay nodes, that distributed it further to the clients. This meant that a lot of users could stream video without overloading the network with a magnitude of individual streams to the one central server.

A peer-to-peer network is a subset of an overlay network[15]. This is a decentralized approach to building overlay networks. Instead of one entity placing nodes at logical locations, as with Overcast, each peer composes the network. Meaning that each peer will both consume and contribute to the network. As described in the 2005 survey[15], there exists two categories of peer-to-peer networks; unstructured and structured. The structure is referring to how the peers structure themselves in the network. In an unstructured network, peers connect and disconnect in an ad hoc fashion. In a structured network there exists some form of hierarchy that the peers must adhere to. Through this structure, a distributed hash table is often formed for many networks. This hash table will allow the peers to cooperate on a common database that is not centrally stored.

Onion routing is another concept closely related to overlay networks[16]. Onion routing is created to offer a way to achieve anonymity in a possibly eavesdropped network. Onion refers to a layered approach to routing the traffic. By having layers with encrypted routing information, it tries to distantiate the originator from the content. To be able to connect the data to the user, you would need to record traffic at all the places it is relayed. Schlegel and Wong[17] have identified that existing anonymising networks, such as Tor, suffer from low bandwidth and higher latency. As for Tor, it can be attributed to how anonymity is achieved through a relay chain and sometimes overworked relays. They also argue that this introduces a degraded usability for the application that these existing solutions actually manage to relay over the network.

2.1.1 Existing hidden overlay networks

A paper from 2007 presents multipath onion routing[18]. In an effort to solve traffic analysis and overloaded relays, the system uses not a single route for the traffic. It distributes traffic across multiple paths, while still preserving anonymity with onion routing. Their own system, MORE, has features which resemble Tor's, but according to them can perform just as good, but with better distribution of the load. Unfortunately this is an entirely new system. In the paper, some active deployment of the system is described. The ability to use multiple paths to the recipient is something we should pursue in our experiments. Which could allow us better performance and combating traffic analysis at the same time.

2.1.2 Anonymous peer-to-peer networks

There exist a plethora of articles that discuss the possibility of anonymous peer-to-peer overlay networks. In one paper by Schlegel in 2012, a new overlay network scheme is presented[5]. They argue that anonymous networks traditionally have had degraded performance, such as low bandwidth, high latency and a somewhat lacking support for routing algorithms. By creating an IP routed network on top of an existing one, they are able to get 50% better performance than existing solutions such as Tor. This overlay IP network is completely disjoint from the substrate network beneath. Some implementation specific problems were identified, such as the need for namespaces and geographical awareness. The latter is for minimizing the delays when relaying the overlay data between substrate routers. They discuss some of the problems when introducing unauthenticated routers in an anonymous network. To combat this, a concept of authenticated routing is introduced in addition to the network. Leaking of network information was unwanted, and kept to a minimum, while packets were able to be relayed to the right router.

A peer-to-peer network, Clouds, was designed and created with anonymity and blocking resistance in mind[6]. Many hurdles, regarding efficiency without directly connecting peers and trust without previously exchanging keys were solved. As with some of the other technologies, it depends on large scale deployment of relays on the internet. However, their principles are said to work on any unstructured peer-to-peer network, which is interesting when regarding our experiment. This means we should be able to deploy their system on top of one we create or facilitate.

A massive actor in the anonymous peer-to-peer network domain is the I2P network[7]. This network started as a fork of another renowned network, Freenet, in 2003. I2P is originally built using many of the same concepts as Tor. These concepts are pivoted toward peer-to-peer setup, compared to how Tor uses them. The most notable difference is how I2P does not use circuits, as Tor does with its Onion routing. Instead, garlic routing is used, which is derived from onion routing. Multiple data segments are encapsulated in a garlic clove, and sent to the routers of the network. In Tor, each data segment is padded to a prefixed length, while in garlic routing these prefixed cells are actually concatenated and distributed. The attribute specifies for how long each router should delay the forwarding of a packet. This, in addition to the delay of accumulating cells in a clove, introduces a certain amount of intended latency.

2.1.3 Defending against traffic analysis

Traffic analysis is the act of collecting data, and only using the metadata associated with the collected traffic[19]. This is often used when the traffic is encrypted, since the eavesdropper can not see the raw unencrypted payload. By using traffic analysis, the eavesdropper can use statistical models to correlate sender and receiver.

An interesting way of defending against traffic analysis was proposed by Beitollahi and Deconinck[20]. By creating a "routing circle" that all the data is sent into, it becomes difficult for an eavesdropping third party. This is due to tha fact that all

the data is relayed in a circular fashion, and not the linear direct link as the standard Tor circuits are. Even though this idea is an interesting take on the traffic analysis problem, it introduces a potential for higher latency. In addition it demands a total refactoring of how circuits are established.

There exists efforts to implement traffic analysis resistance in existing applications, such as BitTorrent[21]. Using a rendezvous point central in the network, they apply a multipath approach. This approach can remind of an enormous mixer that intercepts all streams, and makes it difficult to correlate inward and outward streams. In this way, the data you intend to send or receive gets distributed across many streams before approaching either endpoints. Through this traffic distribution, they achieve high bandwidth, low latency and better traffic analysis resistance.

To reach lower delays in the Tor network M. AlSabah et al. presented a new way of building onion circuits[22]. The idea is to actively having a state over the latency each individual relay contributes. This makes it possible to compose a route which is crafted for the desired latency. As a consequence of this approach, a decrease of anonymity is expected. They tested the system on the Tor network. By preferring other relays than the low bandwidth ones, the system achieved better latency than the original client. Since the circuit throughput is only as fast as the slowest in the chain, one slow relay is enough to make the user experience bad.

Müller performed multiple correlation experiments on the Tor network in 2015[23]. He did not only show how it was possible, but also that it can be defended against with a cost that is acceptable for the Tor network. This is achieved by creating dummy traffic to disturb the traffic analysis sensors.

2.2 Blossom

Due to a fragmented internet, Geoffry Goddel et al. presented Blossom[1]. Blossom is an unstructured peer-to-peer overlay network. It is an effort to create connectivity on a layer on top of the internet. Blossom is treating different logical internet islands as a fragment. These islands are what the technology tries to connect[24]. There are different aspects that define these fragments. Some ISPs block certain content, which this solution would help prevent. Another advantage is that the internet behaves differently, depending on where you originate. This is one of the primary things Blossom tries to mitigate. In the network you are able decide where you originate from when browsing the web. Utilising the Tor network as a substrate, Blossom achieves a worldwide deployment without needing to buy or acquire servers globally. By having these relays in many places, the network is able to circumvent censorship technologies. Content filters, IP address based blocking and network address translation are some technologies it could penetrate.

Architecture wise, it is built as a middleware between the client application and the Tor client. The stack is as follows:

- Client (e.g Firefox)
- Blossom application proxy

- SOCKSifying application (e.g Privoxy)
- Tor Client
- Tor network

In addition to the Blossom application proxy, the software also connects to the Tor client, through a software controller. This controller is what instructs the Tor client to use different relays in the circuit. In its prototype, they utilize the Tor network, not primarily to make it anonymous, but due to the network's extensive reachability. This use of Tor was acknowledged in the anonymity design document by Dingledine and Mathewson in 2006[25].

2.3 Tor

Interest in anonymous routing and information access saw a steady increase during the late 1990s, both in academia and industry. At the Naval Research Laboratory some thoughts was formalized by Paul F. Syverson et al.[16]. In the paper they present a working prototype of what they call "Onion routing". They had created a design for layered packets that travel through a set of proxies before they reach their intended recipients. This worked on multiple protocols, web browsing, remote logins and e-mail. This work created the foundation for the modern anonymity network Tor.

Tor, also referred to as the second generation onion router, was proposed by Roger Dingledine et al. in 2004[26]. The name was originally intended as an abbreviation for "The Onion Router". It was presented as a crowdsourced way of distancing the users of a network with the data they are requesting and producing. It is defined by open specifications[27]. The idea was to relay the data over routers before they exit the network. Each relaying node only observes the previous and next routers, and is only able to decrypt headers that it is intended to relay. Eavesdroppers only observe a part of the chain at a certain location since packages are not taking the shortest path. The Tor network needs a central database over the nodes and the parameters to function properly[28], but having a central actor could lead to compromise of the network. Trust in a single point of failure could lead to a corrupted network. The solution is instead a voting system, where central nodes known as directory servers, agree in cooperation and settles on a consensus document.

Most regular users access the Tor network through the official Tor client[29]. This client is configured in a tested and presumably secure way, making it a preferred way of connecting to the Tor network. It exposes multiple interfaces for incoming connections.

When a user wants to request a resource on the internet, their client first create what is referred to as a Tor circuit. A circuit is, by default, composed of three relays, guard/entry node, middle node and exit node[30]. The intention is to relay the traffic through all of these nodes, with layered encryption, so that each relay only knows the node it was sent from, and where it should be routed to. By doing this, onion routing is able to separate the originator and the information being sent. The last relay, the exit node, is tasked with connecting to the resource on the internet. It is the client of the

network that builds this circuit. The circuit is built one relay at the time. Onion skin is what Tor calls the encryption layer, between each relay. This onion skin consists of cryptographic keys, which needs to be negotiated with each relay in the circuit. Once one skin negotiated, the circuit encrypts all data en uses it to tell the relay to forward the traffic to the next relay it needs to connect to. This enables the client to build as long circuits as needed.

Inside the circuit multiple streams can be operated. These streams bear close resemblance to TCP streams[31]. A difference is that when creating a stream, you can actually either use a domain name or an IP address. The intention behind this is to not leak a DNS requests outside of the Tor session. Tor operates using TCP protocol, and this is not compatible with DNS queries, which is sent over UDP. Therefore, Tor has fixed this with their own stream specification. For the clients of Tor, this happens transparently. SocksV5 is the most common protocol to interface with the Tor client. Socks5 is a protocol for proxying traffic. The clients wrap the data it intends to transmit in a Socks5 session, and sends it to Tor. Both name resolution and what happens beyond this point is not exposed to the client program.

These streams built of Tor cells. These cells are used both to set up streams, establish rendezvous points, but also to relay data. Each relaying data cell is padded to fit 512 bytes; in an effort to not leak data through some forms of traffic analysis.

Latency in the Tor network

Dhungel et al. documented significant latency in the Tor network[32]. Due to Tor's distributed and heterogeneous nature, it varies in the network. Since the Tor protocol also chooses at random which relays it uses, the kind of service you are getting is arbitrary. They showed that at any time, 11% of all Tor routers were overloaded, and introduced delays. These delays fluctuated greatly, but could get as high as a few seconds. With time critical applications, that's not feasible. They also identified guard nodes as a weak point that could introduce a lot of delay due to overwork. They propose making circuits based on delays that the onion proxy has recorded historically. This can introduce circuits that always choose the same router, and sacrificing anonymity.

Palmieri and Pouwelse[33] present a way of distributing the onion routers public keys in a Distributed hash table (DHT) instead of the central server it is today. By doing this they argue that Tor could transition closer to a true peer-to-peer unstructured network. However, a problem arises with this approach. By having an unstructured peer-to-peer network, you no longer know which relays know of each other. They solve this issue by applying a bloom filter[34], so that the paths being chosen do not disclose which relay that know of each other. This is an interesting approach, and makes the Tor network even more blocking resistant, by not maintaining a central directory over all participants in the network.

2.3.1 Hidden services

Hidden service (HS) is the name Tor has given its services that are only available over the Tor network. Traditional use of Tor consists of relaying traffic from the user, over two relays, and exits from the network at an exit node. At this exit node the traffic is completely peeled of onion skin and exposed to the internet. A hidden service is not a service that the exit node can reach by regular measures. The hidden service is reachable through its onion address. This address functions both as its domain name, when postfixed with .onion, and as its key to prove its identity. When you want to reach the hidden service, both parties establish a circuit. The client also establishes a Rendezvous point (RP) at its last relay in the circuit. Through a temporary channel, to an Introduction point, the client signals to the server to connect to the RP.

The current protocol for connecting to hidden services is defined in the "Tor Rendezvous Specification - Version 3"[35]. The intention behind the protocol is that neither the client nor the service are supposed to leak information from the channel. This is enabled by creating two Tor circuits from each end point, and splicing them in the middle. As a consequence, the server is unable to identify the clients, and the clients are unable to identify their service. Since the address also is the key, the users are able to cryptographically verify that the server is the one that possess the address.

When a hidden service is created, the server generates keys and addresses associated with the service. After these keys are generated, it elects some relays to work as introduction points. This list and connection information is encapsulated in a hidden service descriptor and published to the HSDir, also known as the hidden service directory. These introduction points are dedicated relays that coordinate the connection between clients and the hidden service. The introductions points are not responsible for any payload data. When a client wants to connect to the hidden service, it firstly creates a circuit and fetches the descriptor from the HSDir. Afterward it creates a second circuit that it uses to establish the rendezvous point. And finally it creates another circuit that it connects to the introduction point and informs the HS about a pending connection at the rendezvous point. The hidden service connects to the rendezvous point with the parameters received from the introduction point.

An existing chat software, Briar, already utilizes HS for reaching other clients[36]. In Briar you create a HS when you create a user in the peer-to-peer network. The hidden service address is your identifier and is how other users find you when they want initiate a chat session. The Briar network is however limited to chat, forum and blog. No low latency use cases have been implemented. But it is an interesting example of how decentralized communication can be implemented using HS.

2.3.2 Rendezvous points

When Tor connects the HS and client, it utilizes a RP. The RP is unaware of which client and which HS is performing the splice. It only gets a Rendezvous cookie (RC) and patiently holds the circuit alive while waiting for the HS to connect. This cookie is a 20 byte value that uniquely identifies the client which intends to connect. This cookie is sent to the introduction point, so that the HS can connect to the RP.

To enable secure dialogue between two circuits, a relaying entity is needed. This entity is known as the rendezvous point. The overall intention of this functionality is that the rendezvous point should be an unbiased randomly chosen router. The destination of the data is unknown, as the Hidden service name implies. In normal operation it is the client side that initiates the session, by sending an ESTABLISH_RENDEZVOUS_POINT cell to a random selected onion router. This makes the randomly chosen onion router prepare for the job as RP for the session. An important distinction is that it is the client that decides which onion router to use as RP.

The protocol

In 2021 a new specification version of how connections to hidden services in Tor was effectuated in the network[35]. New cryptographic checks were introduced and longer onion addresses were enforced.

A common negotiation is presented (most cryptographic checks are neglected in this example). Alice is the one wanting to connect to a hidden service operated by Bob. This protocol enables a secure way of Alice and Bob to communicate without knowing where or who each other are.

1. Alice establishes a circuit which terminates at an RP with a RC.
2. Alice builds a separate circuit to one of Bob's introduction points, and notifies which RP she has established connection to.
3. Bob builds a 3 hop circuit to Alice's chosen RP and sends a RELAY_COMMAND_RENDEZVOUS1 containing RC and cryptographic keys and digest.
4. If the RP recognizes the RC, it relays the rest of the cell down the corresponding circuit in a RELAY_COMMAND_RENDEZVOUS2 containing the cryptographic keys and digest.
5. Alice receives the RELAY_COMMAND_RENDEZVOUS2, and calculates cryptographic values. If these values do not match the expected, the message is discarded and closes the circuit.
6. Opening stream: Alice sends RELAY_COMMAND_BEGIN with empty address and a chosen port. Bob selects destination IP address and port based on configuration. Then the stream is treated as a regular Tor exit connection.

Challenges to hidden services

When the Hidden service functionality was deployed in 2004, it was the first of its kind at this scale. The ability for both the user and the server to stay hidden was a novel feature at the time. Appraised by organizations working with dissidents and freedom of speech. However, this functionality was vulnerable to intersection attacks, as noted by Øverlier and Syverson[37]. Due to the missing notion of guard nodes, the hidden server would over time leak its location due to the random path it selected. With the guard node functionality it would still be possible to perform the described

attacks, but at a significantly slower rate. An adversary that owns two or more nodes will have the possibility of deanonymizing the hidden service or the user[38]. By not selecting these paths at total random, but with some added functionality, like permanent guard/entry nodes, one minimizes the possibility of a successful attack.

We have now described some central topics in regard to the peer-to-peer structure and anonymous networks. This will be heavily built upon in further chapters, where this knowledge creates the development platform.

Chapter 3

Method

As we have now presented relevant literature, we present how we approach the research questions. This has been separated into three major parts. Each part needs to be solved before the next can be initiated.

To verify both the possibility and properties of using Tor to create a peer-to-peer routable network, we create software that realizes this tunnel. The tunnel software is then used in a sequence of performance tests. These tests intends give a guideline of what is possible to achieve using Tor in the elected configuration.

The developed software creates a network that can be possibly deployed using existing infrastructure, and eventually facilitate solutions as those peer-to-peer networks described in the literature. The literature presents new solutions that demand a critical mass of nodes to operate correctly. By supplying a solution based on the already populated Tor network, we close a gap between Tor and peer-to-peer networking. The solution makes it possible for clients alone to connect and start using the network without the need for volunteers to setup a relay, since Tor already possess a lot of such relays. This enables the use of existing infrastructure to bootstrap and facilitate the next generation of overlay networks. As a consequence of this, we focus more on creating a peer-to-peer platform than connecting fragments, as was done with Blossom.

3.1 Tunnel designs

The Tor network allows for multiple ways to create and design circuits between two clients. Therefore, we create an outline for the different identified ways to create connectivity between two peers. The most suitable configuration, in regard to flexibility, is elected and used to perform the performance tests.

A system where it is possible to configure the number of hops might be preferable, especially in use cases were low latency is prioritized over high anonymity. The user connecting is responsible for using a number of hops that corresponds to its requirements. In a regular Tor circuit, this number is three, but it might be preferential to alternate for different peer-to-peer applications.

3.2 Peer connection designs

After the tunnel mechanism is established, we need to deduce a plan for when and how these tunnels are connected. Before the peers have an active connection, we need a way for them to both find and connect to each other in a decentralized way. This protocol is something each peer needs to know, and is a central part of how the overlay network works. This bootstrapping is needed, so that an overlay network can start without any peers connected. A shared secret is perhaps needed to create initial contact with new peers. As more and more peers connect to the network, these secrets can be forwarded by other peers.

3.3 Performance tests of the chosen design

When the building blocks for the overlay network have been figured out, we execute a series of performance tests. The intention behind these tests is to create an estimate of what the network can deliver. As an indication of what is possible we test the proof of concept software with standardized link tests, extracting both latency and throughput information. In addition, we set up five peers that communicate and forward traffic in the network.

To test performance in a network, the modern tool *iperf3*[39] is a popular option. It is a common tool for testing network throughput, both in academia[40] [41] [42] and in the industry. It is also in use for generating traffic, without the purpose of testing the bandwidth[43]. For testing latency, the native ping command found in Linux is a feasible choice. This command utilizes the ICMP protocol, and it reports back the time the packets used to travel from one address to another.

In this chapter we prepared a method for both creating a prototype and executing valid tests for it. This is the basis for the next chapter when we realize these plans.

Chapter 4

Experiment

In this section we describe the steps needed in designing the channel and peer-to-peer network as a whole. Since we use the Tor network as a substrate we need to initially create a client for this interface. Afterward we describe how the tests will be executed and metrics collected.

4.1 Framework and software

To develop our prototype we need to utilize the Tor network. There exists multiple ways of doing this. One is to utilize the official Tor client. The client for Tor is programmed in C, but forking this could introduce an extended development period. There exists an implementation in Python, named TorPy[44]. This client is both outdated and missing key features, but is a possibly better base than other frameworks. There also exists a framework for orchestrating the original C-client, Stem[45], that was inspected. However, this turned out to be inapplicable to our case, as the original client does not support the granularity high enough for our use case.

The software we developed and used is freely available at:
`https://gitlab.com/fredfall/rtun/`

4.2 Establishing connections between peers

We need to create a tunnel in which we can relay the traffic. There are multiple ways to create the tunnel in Tor, as will be presented shortly. Inside the Tor streams we need to be able to effortlessly send data. To achieve this, an encrypted OpenVPN session is established. This makes it trivial to relay data seamlessly. This is because a native Linux tunneling device is created by the software. OpenVPN also enforces certificates and encryption which we will for simplicity use as the primary encryption and verification between peers.

To be able to build an anonymous overlay network, we need connections between the peers. It is this task that this sections seeks to answer. We need to be able to

place routers in the substrate network that anonymously communicate. We present four different approaches to this, using the Tor network in slightly different ways. The decided solution will dictate how the protocol for connections function. Some preliminary parameters can be assumed exchanged out of bounds. An identifier or something unique of other peers can be assumed. In our experiment we denote these factors as peer1, peer2, and so on. The protocol we develop will also be assumed knowledge, since it is in this frame we establish the channel.

4.2.1 Possible configurations

The official Tor client enables safe and tested options for connecting to the network. Still, by designing our own client we are able to configure tunnels to a greater extent. For the overlay routers to communicate, we need to build a channel for them to communicate directly. In this section we explore different ways of building tunnels in Tor between two peers. P1 and P2 denote the peers that intend to communicate. We start with the shortest possible relay chain, since its requirements give the minimal possible setup.

An important thing to note is that between each official Tor relay there exist a TLS session when information is exchanged. This encrypted stream is wrapped around any data or already encrypted streams going between the relays. If we create a chain of relays, this encryption would add an additional layer. For the clients of the network, this happens transparently and no special steps is needed due to this. Therefore, this TLS layer of security is not illustrated in our figures.

One hop

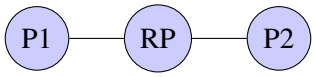


Figure 4.1: Peer P1 and P2 communicates through the rendezvous point RP

One hop configuration is the shortest possible. Where two peers communicate through a single relaying entity. This configuration defeats Tor core principles[27]. The onion routing, which is the heart of Tor, is built on the assumption that the cells are relayed across multiple relays. When only using one hop, many of the implemented mitigations are rendered obsolete. P1(fig. 4.1) is a client, and not a part of the Tor network. The onion router (RP) checks this. Since every public router should exist in the consensus document, it is trivial to get the fingerprint from the connecting router. The reason it does this check is because of denial of service protection. The insecure connections are not the main reason for it. If you could allocate resources with minimal effort, it would be possible to do a massive scale attack with minimal resources. As we will see in the next configuration, it is possible to make this tunnel with two relays. The attack, in this sense, does not scale that massively when you need to pipe it through guard nodes as we will see.

The primary initial reason a one hop configuration is forbidden, is due to the intersection attacks it enables. In the paper discussing this[37], we see how hidden servers could be located by allowing a one hop configuration.

From the relay perspective, one could enable the option "SingleHopRendezvous". This could solve the problem with single relays between peers. A disadvantage is that the relays in the Tor network do not by default have this option set, so we limit our available routers unnecessarily.

The Tor protocol specification[27] also prohibits this use of relays:

```
ORs SHOULD also tear down circuits which attempt to create:
* streams with RELAY_BEGIN, or
* rendezvous points with ESTABLISH_RENDEZVOUS,
   ending at the first hop. Letting Tor be used
   as a single hop proxy makes exit and rendezvous
   nodes a more attractive target for compromise.
```

Because of the potential abuse of resources this could cause and therefor contribute to a DoS attack on its relays.

Two hops

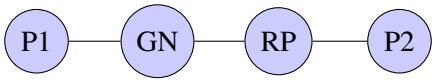


Figure 4.2: Peer P1 and P2 communicate through a guard node, GN, and a rendezvous point, RP

This configuration is similar to the previous one, but an additional guard node is introduced between P1 and the rendezvous point. This makes it possible for P1 to connect to the RP without masquerading as an onion router (since it is technically the guard node that relays the cells that establish the point). In many aspects, this is the same as a one hop. The difference is the forced guard node, which is not possible to remove without further development. An additional node will ensure that the minimal amount of anonymity is better than with only one node between peers.

4.2.2 Three hops, both peers using guard nodes

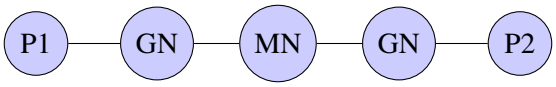


Figure 4.3: Peer P1 and P2 communicate through a guard node, GN, a middle node, MN, and a second guard node.

The RP can check if both the establisher and the connector are onion routers. If it checks both circuits, then the setup in figure 4.3 can mitigate this. By having a guard node before the RP in each directions, the RP is unable to block circuits. As we

discovered, it blocks the establishing part of the RP if it does not originate from an onion router, so it is possible to do this check on the connector side as well.

4.2.3 Four hops, official client through an HS

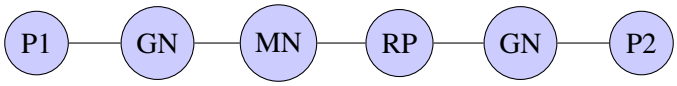


Figure 4.4: Peer P1 and P2 communicate through a guard node, GN, a middle node, MN, a rendezvous point, RP, and a guard node, GN.

Four hops is a minimized version of the official connection used by HS. This altered version is achieved by not using complete circuits at each end. HS originally consists of six hops, two onion routers before the RP on the connector side. The HS connects to the RP with a circuit consisting of three relays, since there exists no trust to the connector chosen RP.

To achieve this four hop circuit, we need to first create a regular hidden service. In the settings, we enable the experimental "one hop"-option. This enables us to use the official client, and no development is necessary. So the connector creates a regular circuit, but the HS-part of the connection only gets one guard node before the RP. This creates some overhead, with introduction points and a second guard node compared to the 2 hop solution. If we use our own encrypted tunnel inside the tunnels, all the onion skin cryptography might lead to unnecessary performance loss. This four hop option implies no major development since it is only a configuration of the client that is necessary.

Utilizing HS functionality in this fashion is something the Briar project [36] does, as mentioned in the literature study.

The selected configuration

In our experiment we intend to use the two hop solution, as described in 4.2.1, as this seems to be a good balance between anonymity and latency. This configuration is also great for expanding a middle node after the guard node, which gives the same number as the Tor official client (three), but avoiding exit nodes.

4.2.4 Splicing the circuits with a rendezvous point

Using the old version 2[46] and new version 3[35] of the Tor rendezvous protocol specification as inspiration, we intend to create a client that can directly use the rendezvous protocol for traffic transmission.

Tor-1 denotes the onion router intended as guard node, and *tor-2* is the router which is used as the RP. Other abbreviations used in the figure for long message names given in the Tor specification are:

- REND_EST - RELAY_COMMAND_RENDEZVOUS_ESTABLISH

- The establishment of an RP using a RELAY_COMMAND type channel.

- REND_ESTB - RELAY_COMMAND_RENDEZVOUS_ESTABLISHED
  - Response from an onion router that the RP is successfully established.
- REND1 - RELAY_COMMAND_RENDEZVOUS1
  - When a hidden service wants to connect to an established RP this type enables the splicing of circuits.
- REND2 - RELAY_COMMAND_RENDEZVOUS2
  - This notifies establisher of an RP that someone have connected the other end of the line.
- RELAY_BEGIN
  - This cell opens a data stream inside the circuit to the other end. This is the same cell type that is used in regular Tor traffic.
- RELAY_CON - RELAY_CONNECTED
  - The response that the other part of a circuit is ready to receive data cells in the stream.

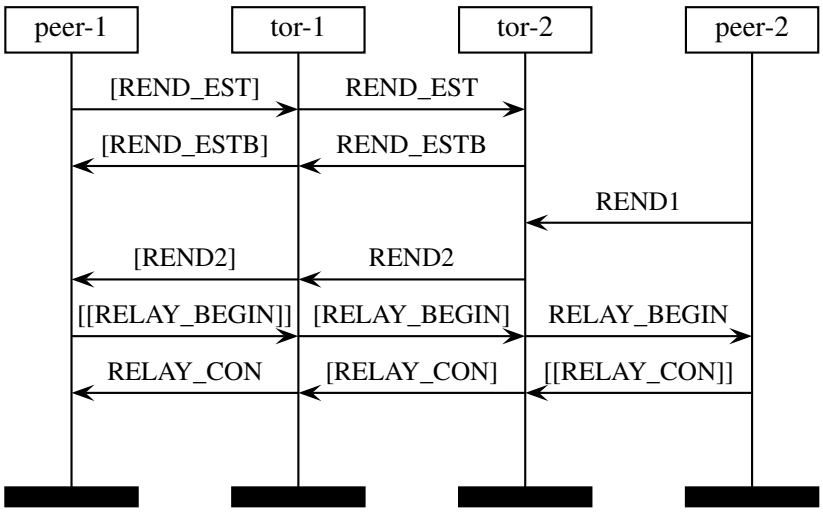


Figure 4.5: The protocol we intend to use to create the tunnel

The square brackets in figure 4.5 denote each layer of onion skin that is applied.

To create a rendezvous point and connect to it, multiple cells need to be encrypted and packaged correctly. The entire session starts with peer-1 creating a TLS session

with tor-1. As mentioned before a TLS session is the base between all Tor relays or clients wanting to communicate. It is not showed in figure 4.5, but it is established between all entities. Peer-1 establishes connection with tor-1 and extends to tor-2. This is done with RELAY_EXTEND cells, and is part of regular Tor circuit establishing. When the circuit is extended to tor-2, we use the onion skin to encrypt a RENDEZVOUS_ESTABLISH packet that is sent to tor-2. Tor-2 establishes the rendezvous point with the attached rendezvous cookie. Consequently, tor-2 sends a confirming RENDEZVOUS_ESTABLISHED cell back to peer-1.

Everything is now set in place for peer-2 to connect to the rendezvous point. It does this by sending a rendezvous cookie packaged in a RENDEZVOUS1 cell to tor-2. Tor-2 responds to this by sending a RENDEZVOUS2 package to peer-1 to notify of the circuit's completion. Peer-1 immediately sends a RELAY_BEGIN cell to tor-2 which relays it. Onion skin between the peers is not established, since there will be an OpenVPN session inside the tunnel. Finally peer-2 responds with a RELAY_CONNECTED, and data can flow between the peers.

Relaying data

In order to relay the data coming from inside the tunnel, the peer-2 needs to relay it. To accomplish this, we mimic how the exit nodes function in the original Tor network. By creating a socket loop that creates new sockets to the intended address when receiving a RELAY_BEGIN with an address and port. In our case, the address is the local interface exposing an OpenVPN service.

Establishing the OpenVPN tunnel

When the RP has spliced the connections, there exists a prolonged onion circuit, which we can send data over. On the initiator side, we expose a SocksV5 interface that relays data to the other side of the tunnel. This mimics the behaviour of the regular Tor client, as described in section 2.3. On this SocksV5 interface we immediately start an OpenVPN session that tries to connect to 127.0.0.1 on the other side. This tries to connect to the loopback interface on the other side of the tunnel. On this loopback interface we expect an OpenVPN listener. The tunnel is presumably able to connect, and a tun-device is created on each side of the tunnel. In our minimal test, all keys and certificates are the same for all sessions.

4.3 Autonomous bootstrapping of the network

Establishing tunnels across the new overlay network demands a scheme for knowing how and where to establish connections to each other. Due to the nature of rendezvous points, both parties in a tunnel setup need to know where and when to establish the initial contact(in addition to a 20 byte cookie). When initial contact is made, it is possible to coordinate further routers to jump to. Since this is not possible before initial contact, we need a plan for where to have the initial RP.

4.3.1 Shared state of available relaying nodes

We need a shared state to derive the chosen RP router from. This shared state needs to be easily available and contain both the same list and sequence, no matter where the request originates.

From this state we can derive selected nodes through a scheme. This scheme needs to always output the same entity. It is imperative for the scheme that the input is the same, in order to produce equal output.

In the Tor network, this shared state is already required by the network, and is known as the Consensus[27]. This document is achieved through a semi-decentralized voting system. Central servers in the Tor network achieve this consensus over a period of time[27]. This list contains a plethora of information, relative to our case. The most relevant for us is the list of all the onion routers that are accepted in the network. This list can be used as an index for all the available RP.

Some preliminary filters is probably a good idea, since not all servers have our preferred properties. If there is a lack of available exit nodes, we do not need to use their limited bandwidth. If network latency plays a crucial part to us, then geographical location could be another filter to apply. The gained anonymity is directly linked to what kind of filter we apply to this list.

4.3.2 Connection schemes

The goal of the scheme is for the two peers to know time and place of connection. There are at least two different ways of doing this. Either by having dedicated plans for each peer pair, or only having a unique plan for connecting to each individual peer. The first we call pairwise, which indicates that each peer pair have their own dedicated scheme for electing a relay to connect to. The second is that each peer have their own scheme for other peers that want to connect to it.

We now present the different ways of connecting to each other using rendezvous points. To clarify, this is before any initial communication has taken place, and is a plot to achieve communication without the need for a central server. In the figures OR is used as an abbreviation for onion router.

Pairwise allocation

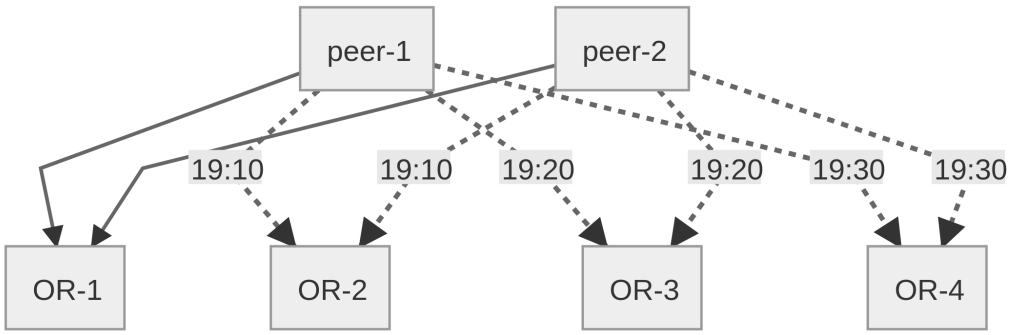


Figure 4.6: Pairwise allocation of RP

Each pair of peers that intend to communicate at some point have their own scheme (e.g if peer x wants to talk to peer y, y or x needs to have a dedicated rendezvous point set up for the other part). Given a continuous connection, this is a great way of always allocating resources. If there exists a lot of peers that are unused, a lot of allocated rendezvous points remain unused.

If the overlay network intends to have all peers connected to each other, you get an enormous increase in the number of rendezvous points needed, as symbolized by figure 4.6. The growth of allocated rendezvous points follows the current format after the third peer:

$$RP_X = RP_{X-1} + p_{X-1}$$

Where p is the number of peers introduced in the network.

It should be predictable where the peer-to-peer connection is taking place. Since both peers have a personalized scheme between them, it is naturally denial of service resistant. Given that only the 2 peers are able to calculate which router to choose as meeting place.

If a user connects to the network with the intention of communicating with its peer, then it tries to connect at the node using the scheme. If no rendezvous point exists, then it is clear that the other part is not connected yet. This means you need to establish a rendezvous point in hope of the other part connecting in the future.

Since each peer can not be sure if it is the first or second to try to establish a connection, they will have to always try to establish a rendezvous point. If this establishment is successful, it means you are the first. If it returns an error, it means the rendezvous is already established. This means the other peer is already connected and that you send a REND1 message and not a REND_EST cell. A probing like this is necessary since it is difficult to know if you are the first to try to initiate contact.

Recipient based allocation

Instead of creating an RP for each peer you intend to communicate with, it is possible to create one RP for everybody that wants to connect to you, but displace the connections in time. This approach assumes that not every peer wants to connect to each other at the same time. The general idea is that each peer that wants an incoming connection, sets up an RP, as shown in figure 4.7. This approach should utilize a faster RP rotation time than the pair approach. The reason for this is that if someone connects to the RP, then it is unable to accept new connections before rotating to a new onion router.

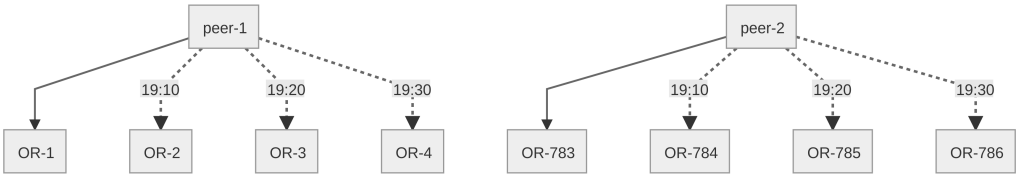


Figure 4.7: Recipient based allocation of RP

To mitigate the problem with RPs, it is possible to employ a scheme which yields multiple possible entry points. By having multiple routers to choose from in each time frame, it is possible to connect to a peer that already has an established connection to someone else. If one router is occupied, it is possible to retry on one of the others routers it should be reachable on.

What is special about this approach is that there is a far less need for allocating rendezvous points. Instead of reserving one for each pair of nodes in the network, you only allocate one for yourself, and hope the other are able to connect to it.

Cookie content

When establishing an RP, you need to supply a 20 byte rendezvous cookie. This cookie is what identifies your channel, and enables that multiple RPs can take place on the same relay. The content of the cookie is something that only the two participating parties should know.

Multiple duplicate cookies on the same onion router throws an error, as seen in the current source code[47]:

Code listing 4.1: Tor source code checks if it is an already existing RENDEZVOUS cookie

```
if (hs_circuitmap_get_rend_circ_relay_side(request)) {
    log_fn(LOG_PROTOCOL_WARN, LD_PROTOCOL,
    "Duplicate␣rendezvous␣cookie␣in␣ESTABLISH_RENDEZVOUS.");
    goto err;
```

Element selection from shared state

In general, the shared state consists primarily of a filtered list, which contain all the current routers we are able to utilize. From this list we need to be able to select the same element from both peers. A general approach is as follows:

Let the length of the list, l, be an integer, and the chosen time resolution be g. If $g = 1hour$, and the current time is 2022-03-02 12:23, we round off to the closest g, in this case 2022-03-02 12:00 and convert it to the Unix timestamp representation. Lastly we summarize the namespace, the logical address of the destination and the timestamp. This summarized value is then hashed and converted to the integer representation of the hash, and finally take the modulo the length l. As time progresses, different rendezvous points are automatically elected for every g. The first 20 bytes of the summarized value is used as the cookie.

In our experiments we simply use the peer names as the tunnel name, so the string representation of this is composed of "peerX"+"peerY". For example peer1 and peer2 become "peer1peer2". Namespaces are meant as a way to avoid collisions where multiple users participate on different networks. In our experiments we use "default" as the namespace. Both the tunnel name and the namespace should be secrets in a real world network, and not predictable. In our experiments we have chosen values which are guessable, and therefore not appropriate in a real world context. If used in the real world, these values should be somewhat random and exchanged before the initial connection.

In our implementation we would end up with something like this:

$$k = \text{"2022-03-02T12:00defaultpeer1peer2"}$$

$$index = int(sha256(k))\%num\_relays$$

It is also possible to design a backup solution, in case the elected node is unavailable or does not respond. One could for example multiply the summarized number with a pre-agreed number, it would produce an offset in the list.

4.4 Performance tests

We will primarily test the network in three scenarios. The first without Tor, consists of only a direct connection between the OpenVPN instances. This is what might be the regular usage today, where the peers are directly connected. A third party could intercept and analyze the traffic going between them. The second is the system we have presented in 4.2.1; OpenVPN through the Tor tunnel consisting of 2 nodes we have control over. The last setup is the same, but with Tor routers which are already existing in the network. Greater latency and throughput is to be expected here.

All experiments utilizing Tor will be conducted with the two hop connection configuration between the peers, as described in 4.2.1.

4.4.1 Latency and throughput

Our prototype could yield a degraded performance, compared to the official client in a regular circuit as mentioned in 4.1. A client to hidden service setup normally involves six relaying nodes with standard configuration. Our proposed system is able to minimally use two nodes, and this is the setup we are intend to test. We sacrifice anonymity for better network properties. The accumulated latency is heavily dependant on the two routers geographical distance and the link capacity between them. The accumulated latency is composed of the four network hops between all the devices. In our test setup, we have the ability to test the latency between each host unbiased by onion routing. This will be used to create a baseline for the tests. By running tests and removing the physical link latency, we get the latency created by Tor and our implementation. To get a best case scenario, we also intend to run OpenVPN without our Tor tunnel. This will be a direct connection between the two nodes, and should inherit a drastically lower latency in comparison.

The throughput can be tested in much the same regard as the latency. It has a closer dependency on the software implementation. Our prototype is as noted implemented in a high level language. The actual maximum bandwidth of the link is not to be expected through our tunnel.

The tool for testing both latency and bandwidth is the iPerf3-tool[39].

An interesting note regarding figure 4.8 is that between GN and RP there exists or is created a TLS connection, which must exist for the two relays to communicate. Over this TLS connection the forwarded data is encrypted and transferred, so there exists a layer on top of what is drafted in the figure.

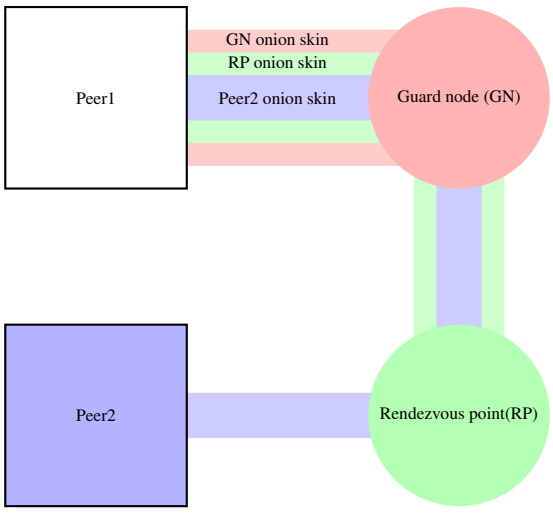


Figure 4.8: How the Tor circuit is built when a tunnel is established

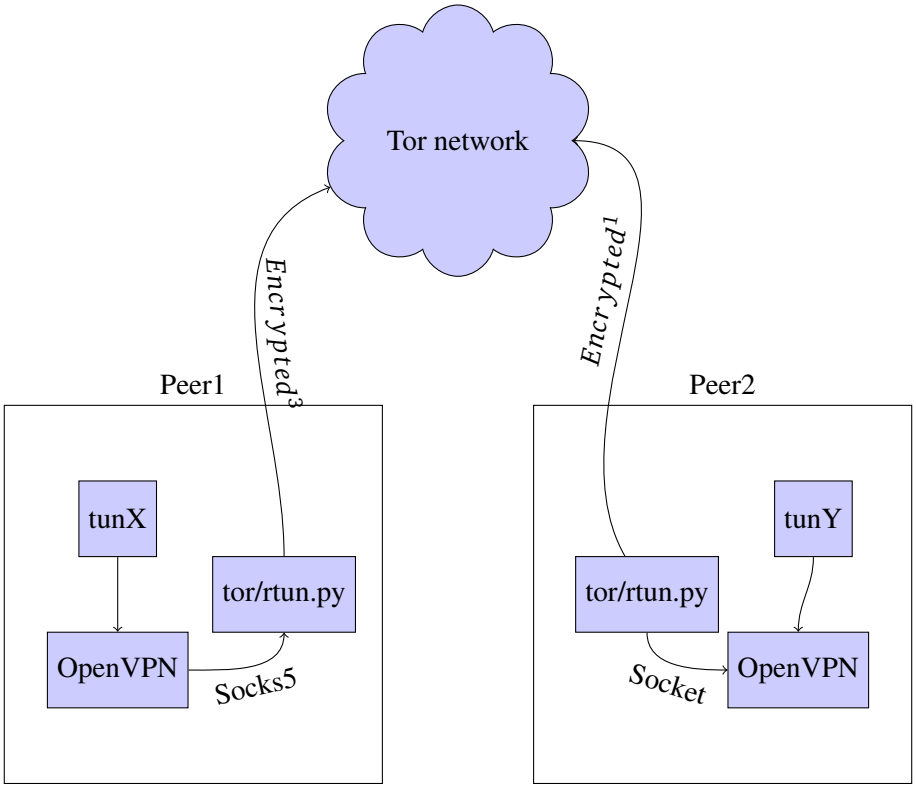


Figure 4.9: Systems on the peers associated with the tunnel

We see in figure 4.9 how the intended designs components interact. Peer1, on the left side, uses Socks5 when connecting through our tunnel software. This terminates at the peer2 tunnel software as a regular socket. The number three on the left side symbolises the minimum number of onion skins.

4.4.2 Routing and resilience

We are going to extend the network by a third node. By having three nodes, we can implement routing strategies, which allows us to build resilience against blocking mechanisms. The intention is that the routers of the created overlay network should be able to route traffic that it not necessarily is connected by a tunnel to.

We intend to connect the nodes as shown below in 4.10:

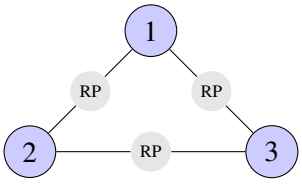


Figure 4.10: Setup of 3 peers with rendezvous points included

Since this is a rather small example, we create a static routing table for each node. In a larger scale, it would be more efficient with a routing protocol that created these tables automatically, but due to limited time time that is deemed out of scope for this thesis.

Routing table for node 1:

```
10.0.2.0/24 via 10.0.0.2 metric 100 # Default for 2 subnet
10.0.3.0/24 via 10.0.0.3 metric 100 # Default for 3 subnet
10.0.2.0/24 via 10.0.0.3 metric 1000 # If link to 2 is down
10.0.3.0/24 via 10.0.0.2 metric 1000 # If link to 3 is down
```

The other nodes will follow the same format, prioritizing direct connections, but having a backup through the other available node.

To test the desired functionality, we will enforce the situation showed in figure 4.11 on the overlay network.

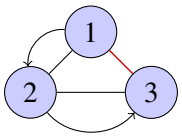


Figure 4.11: The 1-3 connection is disrupted, and data flows through 2 from 1

4.4.3 Scale up to 5 peers

To further investigate the scalability of the network, we intend to test it in a 5 peer network. This 5 peer network will be an expansion of the 3 peer network, as seen in figure 4.12. An intention behind the setup is to create a chain of peers that can forward traffic on behalf of the network. The network need to be able to relay the packets through all the peers, including their respective RP's.

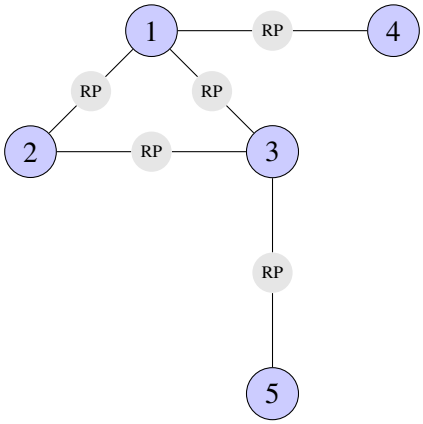


Figure 4.12: Setup of 5 peers with rendezvous points included

4.4.4 Load balancing

In addition to these routing challenges, we intend to shape the traffic to optimize it for certain scenarios. A simple setup of two nodes is sufficient to verify that these technologies can be applied. By having two tunnels connecting the peers at any time, we enable simultaneous use. The first test is to simply distribute the traffic across the tunnels in a round robin fashion, as visualized by figure 4.13. In this way, we may be able to utilize the two tunnels equally. The second solution is to send the traffic asynchronously. This can be done by one peer utilizing one tunnel as upstream link, and the other as a downstream. The counterpart peer will have to be configured in the same way, only mirrored. In figure 4.14 peer1 will use the upmost RP as upstream, and the other as downstream. Peer2 will use the lower as upstream, and the other as downstream.

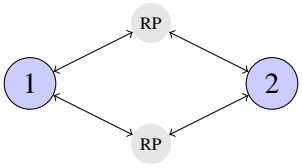


Figure 4.13: Traffic flow of 2 peers with round robin load balancing

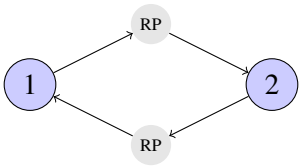


Figure 4.14: Traffic flow of 2 peers with asymmetrical load balancing

Chapter 5

Results

The prototype and experiments gave us interesting results. First we present some design choices regarding the prototype, and then some results from the performance tests.

5.1 Design results

We were able to design and implement a peer-to-peer overlay network system. Through utilizing the rendezvous point functionality in Tor we are able to build tunnels between the peers.

5.1.1 Tunnel

We chose to test on the shortest minimal tunnel configuration. This was the tunnel which allowed the most flexible circuit design. Instead of instrumenting the official Tor client, as was done in the Blossom implementation, we are probably able to chose a path and utilize the network to a greater extent.

5.1.2 Bootstrapping

A consequence of choosing the 2 hop rendezvous approach, there existed a need for deciding which relays to be utilized as rendezvous points. This made it possible to create a fully decentralized bootstrapping mechanism. As described in the previous chapter, we were able to create a fully functioning scheme for choosing a relay out of the consensus. In all our tests we utilize the pairwise bootstrapping mechanism due to the minimal setup. An exception is when we only test a directly connected tunnel.

5.1.3 Prototype

The software we developed and used is freely available at: `https://gitlab.com/fredfall/rtun/`

5.2 Test results

All experiments were conducted on virtual machines running Debian GNU/Linux 11(Linux 5.10.0-13-amd64). The machines were equipped with 4 x Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz (1 Socket) and the other 4 x Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz (1 Socket). Each virtual machine had 1024MB of RAM. Between the two hypervisors, 1GBe ethernet network cards were installed. The software was locked at version:

- ping from iputils 20210202
- iperf 3.9 (cJSON 1.7.13)

Local abbreviations

- T1 - Tor 1 - Our configured relay number 1
- T2 - Tor 2 - Our configured relay number 2
- P1 - Peer 1 - A client/peer in the network utilizing our client
- P2 - Peer 2 - A client/peer in the network utilizing our client

5.3 Latency and throughput

For this test, five virtual machines were utilized, as visualized in figure 5.1. Latency is always stated in milliseconds. A gateway and internet connection is needed to fetch the consensus and descriptors. This is needed since we depend on the live Tor network. Standard deviation is abbreviated to Mdev. T1 will be used as a guard node and T2 as RP. So this setup is the same as the logical figure .

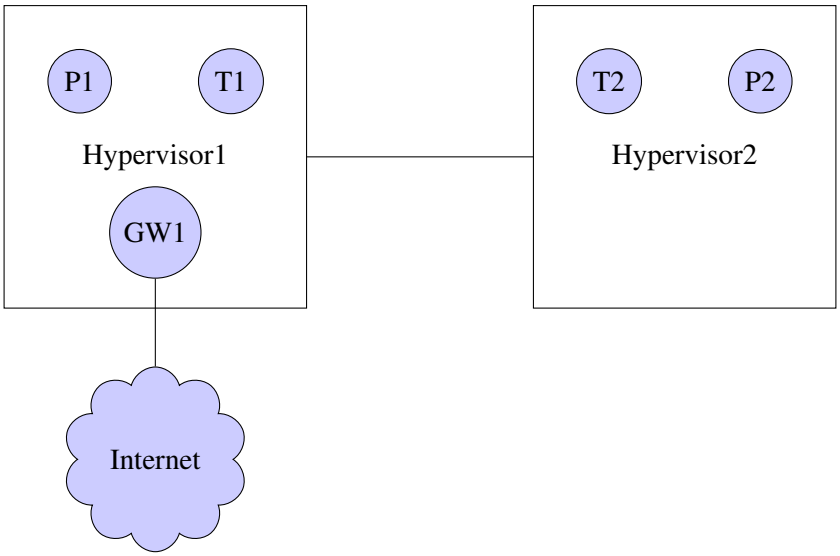


Figure 5.1: Setup for local tests

5.3.1 Results from only using directly connected relays

Between the machines

Table 5.1: Latency(ms) between each individual machine

	T1-P1	T1-T2	T2-P2
Loss	0 %	0 %	0 %
Minimum	0.247	0.587	0.838
Average	0.460	0.644	1.072
Maximum	0.574	0.688	1.272
Mdev	0.115	0.038	0.194

Table 5.2: Throughput between the machines

	T1-P1	T1-T2	T2-P2
From (mbit/s)	2200	934	913
To (mbit/s)	2200	931	911

Peer to peer

Table 5.3: Individual connections between the peers(ms)

	P1-P2	P1-P2 (Ovpn)	P1-T1-T2-P2 (Ovpn+Tor)
Loss	0 %	0 %	0 %
Minimum	0.338	0.609	5.082
Average	0.526	0.887	5.568
Maximum	0.674	1.245	6.060
Mdev	0.123	0.204	0.314

Table 5.4: Throughput between the peers with different tunnel configuration

	P1-P2	P1-P2 (Opvn)	P1-T1-T2-P2 (Ovpn+Tor)
From (mbit/s)	934	123	6.49
To (mbit/s)	937	124	7.22

5.3.2 Test on the live Tor network

We run the live tests in two different configurations. The first is a circuit that only use national nodes. The test was conducted with machines presumably located in Norway. We checked in the Relay search catalogue[48], that each node had at least 30mb/s advertised throughput, so that the network connection of the relay does not choke the Iperf3 test.

For the national setup, these nodes were used:

1. RattyRelay - Norway - C0CBBD17F848C8F9A49104A96DB498013B30F14F - 85.167.143.214:9001
2. PawNetBlue - Norway - 3A9559477D72F71215850C97FA62A0DA7380964B - 185.83.214.69:443

For the international setup:

1. Poiuty - Germany - F6740DEABFD5F62612FA025A5079EA72846B1F67 - 116.202.155.223:443
2. Bigman - United Stated - 519351E3D54202933F85E608D88484A5DC4E4EF0 - 130.61.174.201:9001

In the table 5.5, the tests were executed from the perspective of peer1. So "From" indicates from peer1 to peer2 through the Tor live network.

Table 5.5: Throughput in the live Tor network

	National	International
From (mbit/s)	6.73	6.20
To (mbit/s)	7.71	7.16

Table 5.6: Latency(ms) in the live Tor network

	National	International
Loss	0 %	0 %
Minimum	37.947	86.803
Average	41.439	92.946
Maximum	43.540	113.826
Mdev	2.025	10.463

5.4 Three router setup

These tests are conducted from peer1 to the others. The setup is explained in figure 4.10. The setup starts with all 3 peers being directly connected to each other with

tunnels. After some time, the direct link between peer1 and peer3 is broken, and at a later time reconnected. This test uses the pairwise rendezvous point selection algorithm. This makes it somewhat random which relay is used as rendezvous point.

Table 5.7: Initial latency(ms) in the 3 peer setup

	peer2	peer3
Loss	0 %	0 %
Minimum	69.868	344.485
Average	74.615	352.226
Maximum	81.955	374.347
Mdev	4.017	11.256

Table 5.8: Throughput between the peers in 3 peer setup

	P1-P2	P1-P3	P1-P3 After reconnecting
From (mbit/s)	3.65	3.46	6.00
To (mbit/s)	2.89	2.89	5.00

Table 5.9: Latency(ms) during the tunnel breakdown and layover

	P1 to P3
Loss	2.44 %
Minimum	86.138
Average	731.566
Maximum	6257.795
Mdev	1431.369

The link between peer1 and peer3 has around 90ms when the link is terminated. The latency immediately jumps up to around 250ms, implying that the traffic is now being routed through peer2. The tunnel is then reconnected. During this reconnection phase, the latency skyrockets, as seen in table 5.9, with a maximum packet round trip time of around 6257ms. After some seconds, this instability is eliminated, and the result in table 5.10 is achieved. This session can be further inspected in appendix B.

Table 5.10: Latency(ms) after tunnel reconnects

	peer2	peer3
Loss	0 %	0 %
Minimum	70.364	153.276
Average	71.403	155.408
Maximum	72.619	158.054
Mdev	0.836	1.819

5.5 Five router setup

All tests are executed from peer5. The setup is as portrayed in figure 4.12.

Table 5.11: Latency(ms)

Peer	3	2	1	4
Count	5	5	5	5
Loss	0 %	0 %	0 %	0 %
Minimum	80.093	312.153	125.213	417.433
Average	82.781	341.566	131.674	427.655
Maximum	85.142	410.085	139.957	440.930
Mdev	1.954	35.189	5.844	9.231

Table 5.12: Throughput in the live Tor network

	peer3	2	1	4
From (mbit/s)	2.75	1.70	2.30	1.29
To (mbit/s)	2.05	1.10	1.60	0.77

After 1-3 link is broken

Table 5.13: Latency(ms)

Peer	3	2	1	4
Loss	0 %	0 %	0 %	0 %
Minimum	78.087	299.598	444.126	519.319
Average	82.818	344.295	569.361	576.861
Maximum	88.021	446.835	704.859	627.693
Mdev	3.198	56.332	91.762	39.355

Table 5.14: Throughput having to route through all peers

	3	2	1	4
From (mbit/s)	3.57	1.32	1.20	0.75
To (mbit/s)	2.98	0.93	0.65	0.49

5.6 Load balancing

Table 5.15: Throughput utilizing multiple tunnels between two peers

	Round robin (2 tunnels)	Asynchronous (2 tunnels)
From (mbit/s)	5.10	4.77
To (mbit/s)	4.17	3.87

In 5.15, both load balancing techniques utilized two tunnels. When using round robin, each outgoing packet were transferred on alternating tunnels. When using the other one, we must use two, since upstream and downstream is separated.

Chapter 6

Discussion

We are now done with the results from the experiment, and we have now gathered enough data to make some preliminary conclusions.

6.1 General

Some general factors affected all the experiment. Some of these factors are active choices we make, and others are consequences of how the Tor network is designed.

6.1.1 Testing with unoptimized software

The prototype(`https://gitlab.com/fredfall/rtun/`) is built using technology that are not necessarily built for speed. Python as a language is well known for being slow compared to compiled languages such as C. In addition to this, we use an old framework, creating an unoptimized platform for high speeds and large throughput. However, as stated, the goal of this work was to create a proof of concept. The metrics need to be seen in light of these shortcomings, and only to be taken as an indication of the possibilities.

By using our own implementation of an onion router, we use software that has neither been security reviewed or tested by any external researchers. This makes the software unsuited for real life scenarios that demands any serious privacy requirements. Even if the implementation is based on the principles of the original Tor network, many properties regarding router selection is heavily changed, and potentially creates insecure and short circuits.

6.1.2 Choice of tunnel configuration

We presented multiple tunnel configurations. These were presented ascending by the hops they could support. The two hop solution was chosen, where the initiator first connects to a guard node, before the rendezvous point is established at the next relay. This is the shortest solution available, after the one hop solution was found unfit without further development. This was due to the limitations in the Tor relay software.

By enabling a solution which can be realized by only 2 nodes between peers, a low latency is obtainable. Even though it does not pose as large gain as official Tor circuits, it might be sufficient for some use cases. If we had chosen a three or four hop solution, it would be impossible to achieve the same latency with the same available relays. This is due to the missing ability to configure as few relays in the chain.

We thoroughly investigated the possibility of using only one hop. This, however yielded little results. As one hop allows for intersection attacks and is potential for DoS attack on the network nodes, it is not allowed in the Tor software. There are checks in place to verify whether you are a client or a Tor router. Based on this check the server denies or allows the establishment of RPs from Tor clients (our peers). One can avoid this by having the peers act as Tor relays and not as clients, we can achieve a single hop. This will have a symbiotic effect with the Tor network. We will gain the ability to have a single hop between our peers. And the Tor network will be expanded through all the peers in our network that now are relays. In order to realise this, we need to implement the relay functionality into our clients. This could prove complex, and it might be more realistic to implement our client into a fork of the official Tor client, even though this is a lot of work.

The two hop gives us a more flexible and dynamic approach to generating tunnels, as seen in the two different ways of electing a rendezvous point. This novel way of choosing how to connect would not be possible if we used a three hop regular circuit or a four hop. A major drawback with our solution is that it is dependant on functionality that no software has support for as it requires prototyping. From the performance results it is evident that more development is needed in order to achieve the performance of the regular client. Both the three hop and four hop solution utilizes technology that exists in the Tor network. By doing this, only a little orchestration of the software is needed to create the tunnel. With one of these it would be enough to configure and start Tor, and then start OpenVPN.

In the Blossom implementation the authors were able make use of the regular Tor client. They utilized a control protocol attached to the Tor client. Through this control protocol they orchestrated the client to the needs of the Blossom network. We presented such an approach in 4.2.3. As this would set the minimum of nodes to at least 4, these configurations were not prioritized in our experiments. Unlike our implementation, their intention was to connect fragmented parts of the Internet. As mentioned, our designs are more focused on peers than Blossom, and not aimed at connecting fragments. It is therefore desirable to be able to minimize the number of relays between peers. It should be mentioned that this is not necessarily a recommended setup for all applications, since it becomes vulnerable to surveillance.

The four hop solution (4.2.3) would in addition give a significantly longer establishing time. This proposal was exclusively based on HS functionality. The HS connection protocol is based both on a query to a HSDir and a tunnel negotiation through a introduction node. Compared to the chosen distilled rendezvous connection solution it gains a greater portion of overhead. The 4 hop solution also depended on every peer to host a hidden service that every peer connected to, which is dependant on a selected number of functioning introduction points. The solution we chose

in our experiment is able to utilize all the available relays.

6.1.3 Our tunnel protocol implementation

Since our implementation of the Tor protocol is not as extensive as the official one, some features are missing. Most notably the "SendMe" cell[27], which is used in both directions for traffic management. This is used to either increase the flow of packets or limit it. On the initiator side of the connection, this is actually implemented in TorPy. On the receiver side of the rendezvous tunnel, the "SendMe" functionality is not implemented. This cell is simply ignored in our prototype. Since we have used the Rendezvous specification[35] to establish our protocol, it could be difficult to distinguish our implementation from the original. The exception is the "SendMe" cell that might give our tunnel an abnormal traffic flow.

The Tor software we developed exposes a SOCKSv5 interface, which is what we used OpenVPN to communicate on top of. Standard Tor also exposes SOCKS interface, which the Tor Browser uses to relay traffic. It could be possible to execute the tests directly over this SOCKS interface, and not introduce the encryption layer that OpenVPN adds. However, the applications themselves require support for sending over a SOCKS proxy in that case. OpenVPN removes this requirement, by exposing a native Linux network interface. To shorten the implementation time, OpenVPN was chosen. It creates a native Linux interface which enables us to utilize native routing capabilities of Linux. As a consequence of this, the onion skin between peer1 and peer2 is rendered redundant. In our experiments we only set the onion skin keys to be zeroes, and let OpenVPN have the live encryption keys for the peers. Further development could potentially make OpenVPN superfluous, and only utilize the SOCKSv5 interface and the innermost onion skin.

In the Blossom paper[1] they used privoxy[49] to, as they put it, "SOCKSify" the network traffic. Privoxy creates a SOCKS session for the regular traffic. This has some limitations in regard to which protocols it supports, but it functions well for HTTP traffic. In our implementation we used OpenVPN for this task. This arguably creates more overhead, and introduces more complexity to the tunnels. However, this allowed us to create an overlay IP network over the tunnels with ease. Whereas Privoxy would wrap the traffic in the SOCKS protocol, OpenVPN created a network interface on each peer, that was reachable across our network. Our OpenVPN was set up in peer-mode, meaning it only communicated with another client. It is also possible to configure OpenVPN in a way that allows multiple peers to communicate over the same instance.

Another interesting concept that has emerged in the later years is the concept of cryptokey routing. Presented by Donenfield in 2017[50], the concept is to associate the allowed source IP addresses with a public key of the peer. The convenient consequence of this is that a table with each peer will reveal which encryption key to use for each peer. This corresponds with our decentralized architecture. In addition, this will allow a flexible trust model which creates a peer that can self-configure which peers to trust, and not a central key infrastructure that needs to be trusted.

6.1.4 Bootstrapping

The two different bootstrapping approaches we present come with their own faults and advantages. The one we primarily use in our experiments is the pairwise approach, which scaled sufficiently for our small setup. As mentioned, this require the reservation for separate tunnels for each peer pair. In a setup with many users, this approach could produce a lot of unnecessary overhead. An alternative solution to this could be that when the peer is aware of over 50 peers, a switch is made to the recipient based allocation, which use time as a multiplexer and not the channel.

6.2 Experiment

6.2.1 Directly connected relays

The first test intended to create a latency and throughput maxima. This maximum would create a baseline of what the setup was capable of. The intention was also to eradicate any unforeseen bottlenecks or misconfigurations. As seen in table 5.1, we have an average latency between virtual machines of 1 milliseconds or less, which is to be expected between machines that are almost directly connected. It is strange that T2-P2 connection is the slowest of them all, but this might also be because of differences in hardware. The features extracted from Hypervisor 1 is persistently better than with number 2. However, this is the maximum, and the results from further tests are expected to drop significantly.

The second test is end-to-end connectivity between the two peers. In three stages, we test direct IP link, then direct with OpenVPN, and lastly over OpenVPN in combination with our Tor tunnel. All traffic is only travelling locally in our own network, except for fetching the consensus and descriptors. In table 5.3 and 5.4 we observe the decline in performance as we add these tunneling protocols. Adding the OpenVPN tunnel cause a doubling in terms of latency, probably due to the encryption step. Another reason could be that ping, which uses the ICMP protocol, is tunneled inside a TCP protocol. The throughput is also heavily affected by the tunnel. A tenth of the throughput is achieved. When we add our prototype, we get a latency 5 times as great as that of OpenVPN alone. Lastly, the throughput is a merely 6,5 mbps. The latency can be attributed to the fact that now, the traffic is transiting through 2 extra machines. Meaning it needs to be received by T1, sent from T1, received from T2, and sent from T2. These extra steps are what the 5 millisconds primarily consist of. The small throughput can be attributed to implementation and possibly due to hardware.

In the further tests, we should not expect any higher throughput than 6,5mbps. This is either the maximum for our software, or the onion relays we have configured.

6.2.2 Live two peers on Tor official network

The latency uncovered by the national and international tests were not that surprising, and shows that the distance the packets travel directly impacts the round trip time. For the national, the average latency was around 40 milliseconds, which is not that great

of a leap from the 5 milliseconds when testing locally. The international, where the packets transited Germany and North-America showed over a doubling in latency. This is natural, since the distance traveled is much greater.

A surprising result is that the throughput is relatively persistent in the upper ceiling of what the technology is able to deliver. As shown in the initial test, 6.7, is actually higher than what we were able to achieve using our own hardware. This is an indication that there exists untapped capacity in our implementation. It is not necessarily limited by the latency and distance between relays. It should be noted that we selected Tor relays with high capacity in an effort to not choke the Iperf3-tests. This might not be representative of the network, since the relays usually are automatically chosen, and there exist several slow relays[32].

The rest of the tests use the automatic selection of rendezvous point and guard nodes. This will directly impact both latency and throughout. This is however a representation of how the network actually operates. This is the same flaw as the official client possess. If you choose a path in the network, you automatically decrease the amount of anonymity it gives you. This makes you vulnerable to surveillance attacks over time.

6.2.3 Three router setup

In an effort to create a simple setup that utilized routing, we created a star shaped topology. During the breakdown of the peer1-peer3 direct tunnel, the connection was unstable. In this period we experienced deviations as long as a second on the latency tests. As noted in the result, it was during reconnection that the packets got delayed the most. This may be eradicated by having two tunnels functioning at the same time, switching immediately when a tunnel is disconnected.

An interesting artifact was that when the tunnel got reconnected, the latency was actually lower than when the experiment started. As seen in table 5.10 it is 150ms, while it started with around 350ms. The choice of relay have a drastic effect on the performance.

6.2.4 Five router setup

As an example of a more complex network, we introduce five peers, but unlike the three peer setup, not all peers are connected to each other. This advanced setup underlines that the network is dynamic, and have very few requirements in terms of minimal connectivity. Through the static routes defined at each peer, they are able to route traffic to everybody. As with the previous example, we terminate the peer1-peer3 link. This is forcing all the traffic to be routed through peer2. As we can see in table 5.13, both the latency and throughput have decreased in performance. However, it proves the resilience of the system. It is not dependant on a single link if enough redundancy is established. Each peer can themselves establish extra tunnels, to achieve the redundancy they need. If you know that you are going to exchange a lot of data, it could be an idea to establish a tunnel to that peer. Or even better, multiple, as we will see in 6.2.5.

6.2.5 Load balancing

Two different load balancing techniques were presented. The first was a round robin approach, alternating which rendezvous tunnel when sending data. The second technique utilized two tunnels, using one tunnel for each direction of traffic. Both using multiple tunnels between the same peers to convey traffic. As seen in the table 5.15, this did not yield great results compared to using a single tunnel. This is possibly because we got slow Tor relays. If larger tests were conducted, spanning a large number of relays, this should be possible to investigate further. The same possibly happened with the routing tests in the previous table. If you are unlucky with either the guard or the rendezvous point, throughput can be severely limited.

Another approach to improve the throughput, is to use the approaches described in the literature study[32]. This may be solved by doing preliminary tests when establishing a connection. It is possible to start with 3 initial tunnels, and choose the fastest one of the three after a quick speed test. This is a combination of prioritizing faster relays, but only on a subset of the entire list, and not the entire directory.

A fortunate side effect of trying to achieve higher throughput through load balancing, is that traffic is distributed to more routers. This may in turn make traffic analysis and correlation difficult for a passive surveillance party. If the threat actor only sees a subset of your traffic, then this can make it difficult to detect what kind of traffic you produce. The technique is to send upstream traffic to one rendezvous point and downstream to another. If the threat actor is only able to see one of them, it actively limits the material that can be analyzed. This effect will be increased for each relay included in this setup.

6.3 Security

6.3.1 Routing

A untrusted approach to routing algorithms is to implement it in a way which distributes routing information across all the nodes of the network. If there is absolute trust in the network and the servers that are connected to it, then this is a possible solution. If this network is open to untrusted parties, then leaking routing information is probably unwanted. This can lead to full insight into the entire network. Since we only presented the building blocks of the network, we have not tried to solve this issue. When the namespace is generated, this could be a parameter to declare. If there is trust between each peer, then all routing information can flow freely. In addition each peer decides themselves whether they want to route traffic for other users of the network. In a trusted setup, it would be natural to forward all traffic. But in an untrusted network, one may force the peer to directly connect to the one they want to communicate with. Another reason for not forwarding is that the router does not have a direct route to the intended peer.

In the performance tests, the peers acted as routers when three or more peers were connected. Utilizing a static routing table to forward traffic to the correct peer. As a continuation of this, it is possible to create subnets behind these peers, and treat them

as routers. By doing this, the architecture bears a strong resemblance to the Blossom network. These subnets can be treated as islands that contain a fragmented set of the overlay network, which are connected to the overlay network through the peers. In the Blossom network an important feature was that you could retrieve network services as a function of where you wanted to exit. This setup supports aforementioned functionality if you enable NAT at the peer. The peers could offer access to the internet, but translating your client address when escaping the peer-to-peer overlay network.

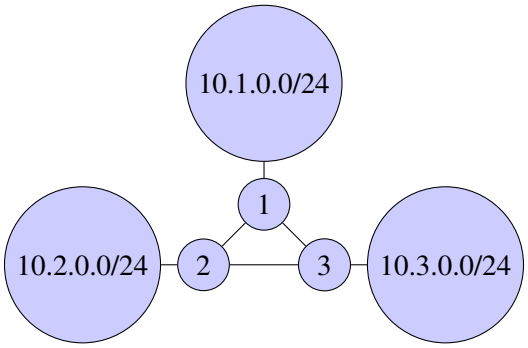


Figure 6.1: Subnets which are accessible behind each peer

6.3.2 Anonymity

No matter where an eavesdropping adversary is located along the Tor circuit, it will know one peer of the connection when only using 2 relaying nodes. This is an effect of our configuration in our experiments. There is no upper limit to how many relays are placed before the rendezvous point. This can be done both from the rendezvous establisher and the connector.

Our overlay network is isolated from the internet. All the parties wanting to participate in the overlay network need to make use of our specially designed Tor gateway. The gateway is technically terminated at the localhost of each peer, which ensures that no clear net traffic can be eavesdropped along the connection.

6.3.3 Anti-Blocking

In a directly connected overlay network, it is possible to block connections between the peers. Effectively terminating part of the transport links of the overlay network. By using our network, it is both harder to detect who is communicating with whom, but it is also impossible to block a single IP address to deter traffic going to the overlay network. Our systems jump between all available Tor relays, which means you either need to block all the nodes in the consensus and keep it updated or do packet inspection and protocol detection as shown in [51].

A way to mitigate this blocking is through the use of Tor Bridges. This existing technology is already in use where Tor relays are blocked in a large scale. By not distributing all the bridges to a single entity, it is possible to grant access to the

network for a majority of the users. Bridges are also possible to use in our setup. Since the bridges are hidden, and not in the central directory, they must be prepended to an existing relay chain if they cannot be verified. They will therefore add a hop independent of the existing planned circuit.

As with HS, the use of rendezvous points enable a natural NAT penetration. This is due to both peers needing to establish a connection outwards from their private networks, to the relay that contains the rendezvous point. This is fortunate for individuals needing to expose a service to the overlay network, without having to configure their home network.

6.3.4 Vulnerabilities and limitations

Vulnerability regarding establishing RP

By first probing to see if the RP is established, we differentiate from the regular Tor rendezvous protocol. This is especially relevant for the pairwise bootstrapping mechanism. The intention behind the functionality is that neither of the peers know if a rendezvous point has already been established. Therefore, there is a need to "probe" the relay for the cookie that the two have agreed upon. This probe will be possible for the rendezvous relay to detect. If the Tor project deems this use of the network unwanted, it can be patched through the detection of this probe. It is therefore possible to block this second connection attempt after the probe. This connection probe can happen during a HS service protocol, if the client loses the connection to the established rendezvous point. In this case, the hidden service will not immediately establish a rendezvous point when the connection fail. Therefore it is possible for the relay to deny establishment of a rendezvous point without destroying existing functionality in Tor.

Predictive rendezvous establishment

If someone is able to figure out the tunnel name, it is possible to establish rendezvous points just before the peers. Again it is the pairwise mechanism that is vulnerable to this kind of attack. This would effectively deny the actual peers to establish a tunnel, since the perpetrator already have reserved it. From the connecting peer this would be perceived as an impersonation of the peer they thought was at the other end. However, since OpenVPN is tunneled inside the rendezvous tunnel, it would decline the certificate and key of the perpetrator. This would thereby only act as a denial of service (DoS) for the peers.

In our experiments we used the tunnel name "peerXpeerY", which is not secure. This is easily guessable for an outsider that knows our protocol. Therefor the tunnel name should be a secret shared between the two peers. Either out of bounds or through a central directory server. Diffie-Hellman could also be utilized, as is done in the original Tor Rendezvous protocol.

No defaults

In our tests we use two jumps, which is few compared to default Tor. This is only for test purposes, and is not a recommended setup for all use cases. Many of the different configurations discussed in this thesis should not be left to the end user. The different parameters should be further tested and configurations should be created by the application that utilizes the overlay network. A use case is to transfer a file to another peer. If there exists no time requirements, it is possible to send it through 6 relays and with a capped bandwidth. Either for the reason of deterring traffic analysis or simply to avoid a strain on the Tor network.

6.3.5 Block unwanted RP use

The Tor project could identify this use of rendezvous points unwanted, either due to security issues or unwanted traffic. If this is the case, it could be possible to demand that you verify yourself as a HS when connecting to a rendezvous point. The establisher of rendezvous points are users, so it would not be possible to check it at this end. When the hidden service connects to the rendezvous point, it would be possible to demand some sort of authentication. However, this could also prove difficult, since the HS should not at any point disclose which HS it is to the rendezvous point. Any use of keys associated with an onion address is unwanted. Such a check would also violate the principle of having out-of-bounds HS not published. Tor cannot do this without reducing already existing features.

6.3.6 Traffic analysis

Due to the decentralized nature of our architecture, intersection attacks will not be possible to effectively utilize. With the hidden service attack, this was possible to initiate since one can personally connect to the service. This creates circuits that are possible to analyse. In our setup there will exist circuits, but these are only possible to initiate from the peers that have the common secret. In our experiment, this secret were derived from the peer names and not a representative example of a secure configuration.

6.4 Optimizations

6.4.1 Organic network expansion

We experienced in the 5 peer setup that both the latency and throughput were limited when packets needed to be forwarded through 3 peers. In this case, peer5 should have established a direct connection with peer4. This conclusion may the network be able to make, and thereby automatically connect to peers it wants to communicate with. This can for example be because of a need to communicate over prolonged time or in such a quantity it puts unnecessary strain on the forwarding peers. It may use other peers during the connection phase, to get a relatively good connection quick,

and switch when the new tunnel is established. This demand driven tunnel generation could help lessen the strain on the network. To have multiple routes would also help build redundancy.

6.4.2 RP jumping during an ongoing session

As we discussed in 4.3.2, we utilize the different relays when waiting for a connection to the RP. It is also possible to do this while the tunnel is established. Say we want to change routes every 30 minutes to combat traffic analysis. It could be possible to send a special packet that is interpreted by the created Tor client. In this packet the next onion router to be used as RP could be defined, as well as when the switch should be made. This can enable a seamless transition to the new RP, without the OpenVPN tunnel detecting it. This would be especially good for recipient base allocation, since there does not exist a relay plan between the peers.

6.4.3 Circular data flow

It is possible to only send data in one direction in a circular setup of the network, as described in [20]. This will mitigate traffic analysis through receiving and forwarding traffic not intended for you in the circle network. A solution like this is not based on multiple paths, but everyone using the same path. The peers get organized in circle, and forwards all traffic, no matter who is the recipient, to the peer to its left. This configuration is achieved by defining the peer to the left as your default gateway. If every peer has this configured, it will be sent in a circle. The last peer must define peer1 as its default gateway to terminate the circle. Traffic that has you as a destination is not forwarded. It should be noted that this will produce large delays, depending on the size and quality of network.

6.4.4 Directory servers

Even though we prioritized decentralized solutions, one could possibly have a central coordinating server in the peer-to-peer network. This is particularly helpful to discover other peers. This server could listen on the same address in all namespaces. The server either has a logical address or a name for all the routers that are accessible in the network. Another approach is that this directory server could pair you with the router you intend to communicate with. This is helpful when contact information has not been exchanged prior to the first contact. Both of these solutions are not decentralized and are vulnerable to attacks against the directory server which is something we do not want.

Chapter 7

Conclusion

We managed to realize and test a peer-to-peer overlay network utilizing Tor relays. By creating specially crafted circuits that utilize the rendezvous functionality in Tor, we are able to create a fully decentralized bootstrapping mechanism for how the peers connect to each other. This solution was selected after we investigated multiple ways of creating tunnels between peers in a peer-to-peer structure. Due to the nature of rendezvous points, we were able to bootstrap tunnels with few prior parameters.

We also show how data can be forwarded, minimizing the need for a tunnel between all peers in the network. In addition, multiple tunnels between peers can be utilized to possibly achieve higher throughput. This flexible use of tunnels and forwarding can be used to further combat traffic analysis, by utilizing different routes to the destination. In addition, the decentralized nature of both our solution and the Tor network provides the network with built-in blocking resistance. With time, all the relays available in the consensus will be utilized, skipping relays that are blocked or unavailable. In this fashion, the network avoids having a single point of failure in both network and available services.

Our implementation did not manage to utilize the relays to their full potential, with a possible software bottleneck. This is probably due to the implementation, and not because of limitations in the Tor network. Although the performance degraded rapidly when introducing long paths in the topology, we presented methods by which the peers can mitigate this. The performance is directly connected to geographical location and available bandwidth of the onion relays.

Chapter 8

Future work

It is possible to expand this network in different directions, due to the dynamic interfacing capabilities. An interesting direction is to deploy a peer-to-peer application on top of this network, for example BitTorrent[52], Skype[53] or InterPlanetary file system (IPFS)[54]. Utilizing both the tunnels we presented in our paper and utilizing the routing capabilities of the network. This enables data flow without each peer being connected to each other.

Due to the limited bandwidth we were able to get from our prototype, it could be interesting to utilize the official client for better performance. If it is possible to use their library for tunnel performance and better protocol implementation, this could yield a better utilization of the Tor network. This could yield a more stable implementation. Better parsing of Tor cells and error handling is to be expected.

Another direction is to conduct a security review into how a relay chain can be considered anonymous in a peer-to-peer setup. By combining research into peer-to-peer network with our prototype, it could be possible to establish a configuration which is suitable for a decentralized network. In this review, a special focus on anonymity should be prioritized. The review can build upon existing research and proofs associated with the Tor project.

Bibliography

[1] G. L. Goodell, S. O. Bradner and M. Roussopoulos, 'Blossom: A decentralized approach to overcoming systemic internet fragmentation,' *FAS Scholarly Articles in the Harvard Library office of Scholarly Communication*, 2005.

[2] B. N. Levine, M. K. Reiter, C. Wang and M. Wright, 'Timing attacks in low-latency mix systems,' in *International Conference on Financial Cryptography*, Springer, 2004, pp. 251–265.

[3] S. Chakravarty, A. Stavrou and A. D. Keromytis, 'Traffic analysis against low-latency anonymity networks using available bandwidth estimation,' in *European symposium on research in computer security*, Springer, 2010, pp. 249–267.

[4] P. Mittal, A. Khurshid, J. Juen, M. Caesar and N. Borisov, 'Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting,' in *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011, pp. 215–226.

[5] R. Schlegel and D. S. Wong, 'Anonymous overlay network supporting authenticated routing,' *Information Sciences*, vol. 210, pp. 99–117, 2012.

[6] M. Backes, M. Hamerlik, A. Linari, M. Maffei, C. Tryfonopoulos and G. Weikum, 'Anonymity and censorship resistance in unstructured overlay networks,' in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2009, pp. 147–164.

[7] B. Zantout, R. Haraty *et al.*, 'I2p data communication system,' in *Proceedings of ICN*, Citeseer, 2011, pp. 401–409.

[8] J. Oikarinen and D. Reed, 'Internet relay chat protocol,' *Network Working Group at the Internet Engineering Task Force*, 1993.

[9] K. Abe and S. Goto, 'Fingerprinting attack on tor anonymity using deep learning,' *Proceedings of the Asia-Pacific Advanced Network*, vol. 42, pp. 15–20, 2016.

[10] A. Panchenko, F. Lanze and T. Engel, 'Improving performance and anonymity in the tor network,' in *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2012, pp. 1–10.

[11] R. Radner, 'The organization of decentralized information processing,' *Econometrica: Journal of the Econometric Society*, pp. 1109–1146, 1993.

[12] C. Diaz, S. Seys, J. Claessens and B. Preneel, 'Towards measuring anonymity,' in *International Workshop on Privacy Enhancing Technologies*, Springer, 2002, pp. 54–68.

[13] J. Buxton and T. Bingham, 'The rise and challenge of dark net drug markets,' *Policy brief*, vol. 7, no. 2, pp. 1–24, 2015.

[14] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek and J. W. O'Toole Jr, 'Overcast: Reliable multicasting with an overlay network,' in *Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.

[15] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma and S. Lim, 'A survey and comparison of peer-to-peer overlay network schemes,' *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[16] M. G. Reed, P. F. Syverson and D. M. Goldschlag, 'Anonymous connections and onion routing,' *IEEE Journal on Selected areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.

[17] R. Schlegel and D. S. Wong, 'Low latency high bandwidth anonymous overlay network with anonymous routing,' *Cryptology ePrint Archive*, 2009.

[18] O. Landsiedel, A. Pimenidis, K. Wehrle, H. Niedermayer and G. Carle, 'Dynamic multipath onion routing in anonymous peer-to-peer overlay networks,' in *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, IEEE, 2007, pp. 64–69.

[19] S. J. Murdoch and G. Danezis, 'Low-cost traffic analysis of tor,' in *2005 IEEE Symposium on Security and Privacy (S&P'05)*, IEEE, 2005, pp. 183–195.

[20] H. Beitollahi and G. Deconinck, 'Ferris wheel: A ring based onion circuit for hidden services,' *Computer Communications*, vol. 35, no. 7, pp. 829–841, 2012.

[21] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani and P. Francis, 'Towards efficient traffic-analysis resistant anonymity networks,' *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 303–314, 2013.

[22] M. AlSabah, K. Bauer, T. Elahi and I. Goldberg, 'The path less travelled: Overcoming tors bottlenecks with multipaths,' *University of Waterloo Centre for Applied Cryptographic Research, Technical Report CACR*, vol. 29, 2011.

[23] K. Müller, 'Defending end-to-end confirmation attacks against the tor network,' M.S. thesis, 2015.

[24] G. Goodell, S. Bradner and M. Roussopoulos, 'Building a coreless internet without ripping out the core,' in *Fourth Workshop on Hot Topics in Networks*, Citeseer, 2005.

[25] R. Dingledine and N. Mathewson, 'Design of a blocking-resistant anonymity system,' *Technical design document published at Freehaven.net*, 2006.

[26] R. Dingledine, N. Mathewson and P. Syverson, 'Tor: The second-generation onion router,' Naval Research Lab Washington DC, Tech. Rep., 2004.

[27] *tor-spec.txt - torspec - Tor's protocol specifications*, [Online; accessed 5. Apr. 2022], Apr. 2022. [Online]. Available: `https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt`.

[28] A. Biryukov, I. Pustogarov, F. Thill and R.-P. Weinmann, 'Content and popularity analysis of tor hidden services,' in *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, 2014, pp. 188–193.

[29] *The Tor Project | Privacy & Freedom Online*, [Online; accessed 16. May 2022], May 2022. [Online]. Available: `https://www.torproject.org/download`.

[30] A. Kate and I. Goldberg, 'Using sphinx to improve onion routing circuit construction,' in *International Conference on Financial Cryptography and Data Security*, Springer, 2010, pp. 359–366.

[31] R. A. Haraty and B. Zantout, 'The tor data communication system: A survey,' in *2014 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2014, pp. 1–6.

[32] P. Dhungel, M. Steiner, I. Rimac, V. Hilt and K. W. Ross, 'Waiting for anonymity: Understanding delays in the tor overlay,' in *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, IEEE, 2010, pp. 1–4.

[33] P. Palmieri and J. Pouwelse, 'Key management for onion routing in a true peer to peer setting,' in *International Workshop on Security*, Springer, 2014, pp. 62–71.

[34] S. Geravand and M. Ahmadi, 'Bloom filter applications in network security: A state-of-the-art survey,' *Computer Networks*, vol. 57, no. 18, pp. 4047–4064, 2013.

[35] *rend-spec-v3.txt - torspec - Tor's protocol specifications*, [Online; accessed 5. Apr. 2022], Apr. 2022. [Online]. Available: `https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt`.

[36] *Secure messaging, anywhere - Briar*, [Online; accessed 15. May 2022], May 2022. [Online]. Available: `https://briarproject.org`.

[37] L. Overlier and P. Syverson, 'Locating hidden servers,' in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, IEEE, 2006, 15–pp.

[38] P. Syverson, G. Tsudik, M. Reed and C. Landwehr, 'Towards an analysis of onion routing security,' in *Designing Privacy Enhancing Technologies*, Springer, 2001, pp. 96–114.

[39] V. Gueant, *iPerf - The TCP, UDP and SCTP network bandwidth measurement tool*, [Online; accessed 9. May 2022], May 2022. [Online]. Available: `https://iperf.fr`.

[40] S. Bhandarkar, S. Jain and A. N. Reddy, 'Ltcp: Improving the performance of tcp in highspeed networks,' *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 41–50, 2006.

[41] H. Sivakumar, S. Bailey and R. L. Grossman, 'Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks,' in *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, IEEE, 2000, pp. 38–38.

[42] V. J. D. Barayuga and W. E. S. Yu, 'Packet level tcp performance of nat44, nat64 and ipv6 using iperf in the context of ipv6 migration,' in *2015 5th International Conference on IT Convergence and Security (ICITCS)*, IEEE, 2015, pp. 1–3.

[43] S. Srivastava, S. Anmulwar, A. Sapkal, T. Batra, A. K. Gupta and V. Kumar, 'Comparative study of various traffic generator tools,' in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, IEEE, 2014, pp. 1–6.

[44] torpyorg, *torpy*, [Online; accessed 19. May 2022], May 2022. [Online]. Available: `https://github.com/torpyorg/torpy`.

[45] *Welcome to Stem! Stem 1.8.0 documentation*, [Online; accessed 14. May 2022], Sep. 2020. [Online]. Available: `https://stem.torproject.org`.

[46] *rend-spec-v2.txt - torspec - Tor's protocol specifications*, [Online; accessed 5. Apr. 2022], Apr. 2022. [Online]. Available: `https://gitweb.torproject.org/torspec.git/tree/rend-spec-v2.txt`.

[47] *src/feature/rend/rendmid.c · main · The Tor Project / Core / Tor*, [Online; accessed 16. May 2022], May 2022. [Online]. Available: `https://gitlab.torproject.org/tpo/core/tor/-/blob/main/src/feature/rend/rendmid.c`.

[48] *Relay Search*, [Online; accessed 25. Apr. 2022], Apr. 2022. [Online]. Available: `https://metrics.torproject.org/rs.html#`.

[49] *Privoxy - Home Page*, [Online; accessed 13. May 2022], Jan. 2022. [Online]. Available: `https://www.privoxy.org`.

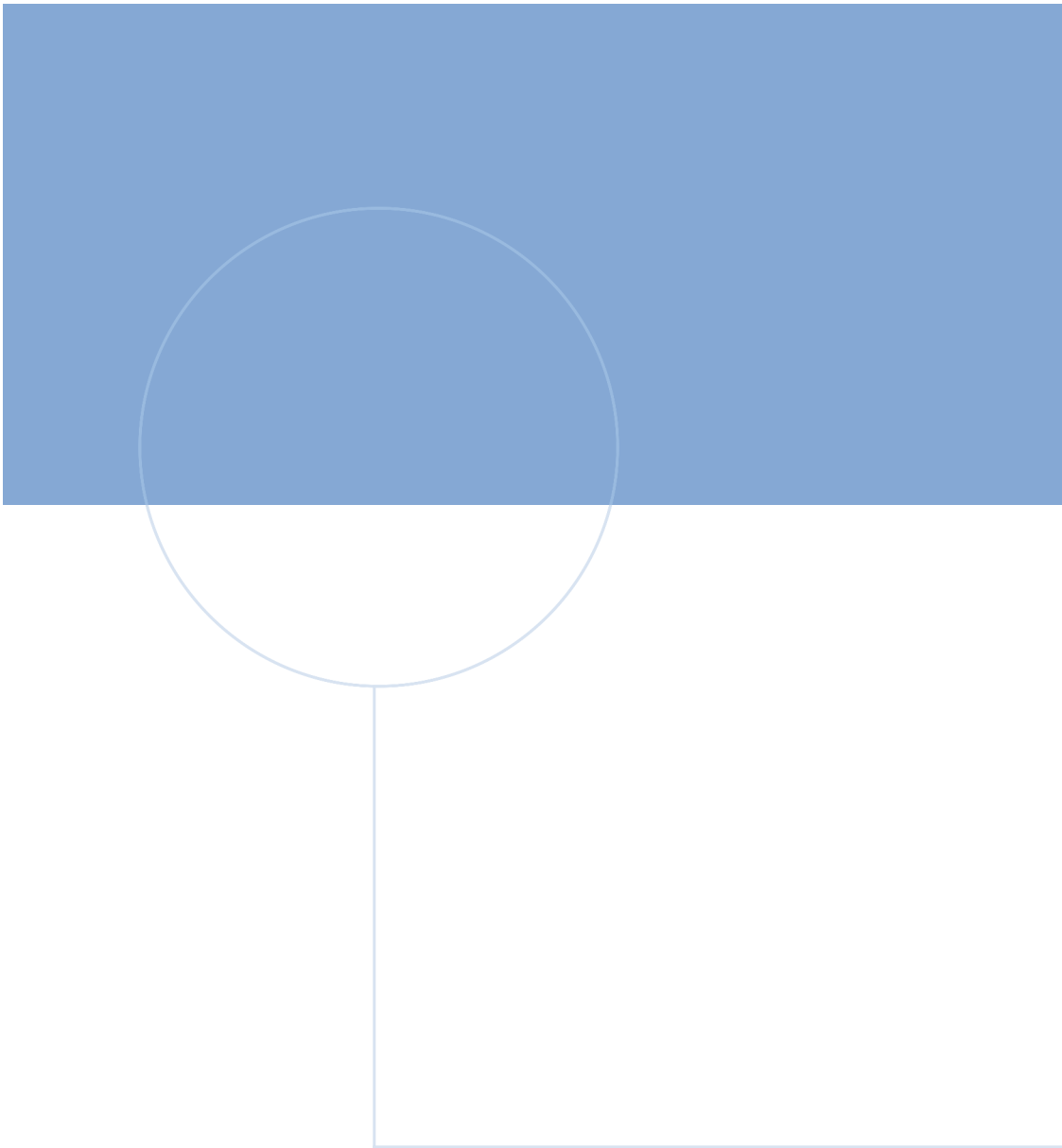
[50] J. A. Donenfeld, 'Wireguard: Next generation kernel network tunnel.,' in *NDSS*, 2017, pp. 1–12.

[51] A. Borge and F. Fallang, *Detection of malicious software implants that enable TOR, The Onion Router, for anonymization*, [Bachelor thesis], 2016.

[52] B. Cohen, 'Incentives build robustness in bittorrent,' in *Workshop on Economics of Peer-to-Peer systems*, Berkeley, CA, USA, vol. 6, 2003, pp. 68–72.

[53] *Skype | Hold kontakten med kostnadsfrie videosamtaler over hele verden*, [Online; accessed 16. May 2022], May 2022. [Online]. Available: `https://www.skype.com/no`.

[54] *IPFS Powers the Distributed Web*, [Online; accessed 16. May 2022], May 2022. [Online]. Available: `https://ipfs.io`.



NTNU

Norwegian University of
Science and Technology