UNIVERSITY OF COPENHAGEN

Organized Peer-to-Peer Network

Enabling interconnectivity and distributing workload

Mads Ynddal
SJT402

June 13, 2016

Abstract

Accessing devices on a local network has become challenging with the development of the Internet. Recent shifts to cloud-based services, has overseen the need to access these devices, such as printer and cameras. Apart from accessing them, doing this in an efficient manner is also a challenge. This project proposes a peer-to-peer based solution, which will allow for communicating with a local device through a cloud-service. Because of the nature of the peer-to-peer system, it will also allows for high performance gains and reduced operation costs by distributing workload.

Forewords

This project was inspired by an obstacle I encountered at my current place of employment. The company is making print-managment software and wanted to move from a traditional on-premise server to a cloud-based solution. If possible, this would make maintenance easier for all parties and reduce the cost of the system. Having a central server, would also allow for quicker and more fluent updates.

Eventhough the company is working on their own system, they allowed me to work with the idea and see where it would lead. My solution incorporates a peer-to-peer network, which will allow for greater flexibility and performance, but at the cost of a slightly more complicated system.

Contents

Category
Peer-to-peer

Keywords
Copenhagen, university, computer, science, P2P, peer, peer-to-peer, node, super, cloud

1 Introduction

The central goal of this project, is to make registered devices available from anywhere on the Internet, without special network configurations. On top of this, explore if it is possible to do this in an efficient way, where a single server is not having the burden of proxying all of the connections to these devices.

On the Internet today, we have to look out for hackers and protect our privacy. Because of this, we cannot normally access LAN-devices from the Internet. But what do we do, if that is what we want?

1.1 Background

Before designing the system, we have to look at the challenges we face, with the current technology. The world has changed a lot since the inception of the Internet.

1.1.1 Cloud Services

In the recent shift to cloud solutions, the focus has been on reducing the workload for the local administrators and reduce on-premise support. And it has done so very well and with great benefits[19].

In this progress, it has been overseen, how we would solve scenarios, where devices on a local network, are inaccessible from the outside world. Our current view of cloud services often involves an external server that is accessed through a webpage. On these setups, it is not a possiblity to access a device on a local network. If we wanted access from the cloud, we would need a specialized server that could facilitate the requests on behalf of the cloud system.

1.1.2 On-Premise

Traditionally, each company had a computer dedicated to be a server, whether it was on a server rack or a workstation in a broom closet. The disadvantages to this, is the need to reconfigurate the network's forwarding and firewall rules and general maintenance of running a server. This require the company to have an expensive consultant or full-time employee to work on it. This can be challenging for a small or medium sized business.

1.1.3 End-to-end Internet

The main goal is to access a LAN-connected device from anywhere on the internet. This is commonly not possible, because it opens the network to a wide range of attacks by hackers or general abuse.

When the Internet was invented, it was the intention, that every computer would have its own IP address[21]. In this time, any person could sit down and configure their computer to host a webpage or let any connected computer be accessed from anywhere.

This time ended, as we began to expand the Internet and the Internet Service Providers (ISP) only provided one IP for each subscriber (with the exception being dedicated server hosts). This meant, that techologies like Network Address Translation (NAT) had to be used, which helps "splitting" the Internet between multiple computers. I will get into how a NAT works in section 1.3.1.

The result of this development meant, that it has become harder to start a server, as you need access to configure the NAT inside the local router. This is to some degree mitigated by the use of UPnP, which I will get into in section 3.4.3.

In the near future, the use of IPv6 could solve some of these issues, as IPv6 gives makes it possible for every computer to get its own IP[11, introduction]. The current Internet primarily uses IPv4, which, without respect to special addresses, supports 2^{32} (4 bytes) unique address or equivilant to approximately 4.3×10^9. These addresses

has all been used and we will be forced to increase the address space soon. The IPv6 standard has increased the address space to 2^{128} (16 bytes), which has $3,4 \times 10^{38}$ unique address. The amount of addresses in IPv6 is so tremendously vast, that some have speculated, that every grain of sand on the planet, could be addressed uniquely, and there would still be addresses left [2].

Although it's hard to predict the future, I speculate, that IPv6 might not mitigate the problem at hand. We have grown too accustomed to the idea, that our computers are not accessible from the Internet. It started as a restriction, but nowadays, it might be a good way to hide a computer with vulnerable software. Or to hide from adware-tracking, by sharing an IP with other people.

The point of this project will be to regain this kind of access in a secure manner, without exposing every device to the world.

1.2 Case

This project comes from a usecase, of accessing a local device, from anywhere on the Internet. There are several devices, that could utilize this kind of connectivity, but is limited by the current technology of routers, NATs and so on.

I've constructed three cases, to show the motivation for this project. The project is not going to solve the whole case, but could be used to connect devices, that a third-party application then uses.

1.2.1 Printers

The first case is the fictitious company Printer Corp., which sells printers and print-management software. They have around 100 companies as customers, who each own 10 to 50 printers. These printers are often spread out between multiple buildings within the companies. The companies manage their printers through a local server, which Printer Corp. helps setting up and maintain.

Printer Corp. wants to move their business into the cloud, to ease the maintenance and reduce on-site support. But they foresee difficulities in connecting to the local printers. These printers are most often on an isolated network by themselves or on a local subnet with users.

1.2.2 Security Cameras

The second case is the fictitious company Vision Security, which specializes in selling and setting up security cameras. The company wants to ease the set up of cameras, as this often involves a server that has to be set up on-premise to give the user access to a UI to control the cameras.

These cameras are most beneficial to access from the Internet, as cameras are the most ideal when you are not close to them. Therefore, it has to be accesible from anywhere on the Internet.

Installing the camera is a tedious task, as an employee has to physically show up and help, in setting it up. It also involves cooperating with the customer's network-administrator, which they might not have or is not a full-time employee.

They want a solution, which is just plug-n-play: Mount the camera onto a wall, connect to power, connect to the Internet and it will by itself figure out the rest.

They also want to it be set up in a way that does not directly allow access to the device for everybody. At the current time, some companies expose these cameras to the world without a password or unknowingly installed a camera, which has a known vulnerability [12].

1.2.3 Internet of things

People have speculated, that anything has to be connected to the Internet in the future, even your toaster. This would mean, that these devices possibly has to be accessible from anywhere in the world.

As the current way routers work, it will require a relatively high level of skills, to make these devices really work.

But the company making these devices could also foresee this, and implement something in the lines of this project, to make the connections.

1.3 Obstacles

As lightly described in the previous section, we have a few obstacles when it comes to make devices available from the Internet. This section will give a brief background on the issue, and we will try to solve them in section 3.4.

1.3.1 NAT

One of the most common techonlogies on the Internet today, is the Network Address Translation (NAT)[16]. It enables multiple computers to be sharing the same IP address. Meaning, that if you live with 5 people in the same house and visit `http://www.google.com`, Google would not, based on the IP, be able to differentiate between the 5 connections as different computers.

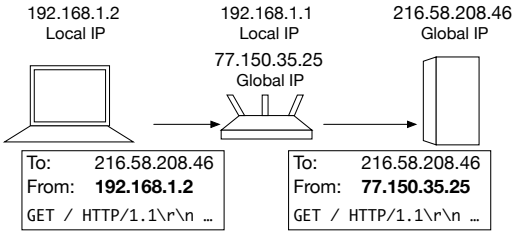

Figure 1: Simplified example of NAT

Illustrated by figure 1, the user's computer on `192.168.1.2` sends a request to `http://www.google.com` (on IP 216.58.208.46). It is received by the router, which makes an entry in its internal NAT-table. The router then changes the IP of where the request should be returned to, and sends it off to Google. When Google receives the request, it has no way to determine, that the request has been altered by the router, and returns it to the IP it has been given. When the request gets received by the router, it looks in the table for whom requested something from Google, changes the IP back again and sends it to that local computer. I've omitted the ports from the illustration to make the explanation simpler, but these take part in differentiating between multiple connection to Google.

The NAT has the side-effect, that if you are trying to host a server, and a connection comes into the router from the Internet, it has no idea which of the multiple local computers, that should be receiving the packet, if there is no entry in the NAT. And as destribed above, entries in the NAT are only made on outbound connections.

There exists a few ways to make the NAT accept incoming connections and redirect them to the appropriate host. We look at these options in section 3.4.

1.3.2 Firewall

Some networks have a dedicated appliances called a firewall. Its job is to monitor and control data that is being transferred locally or being sent or received from the Internet.

The firewall incorporates a set of rules, which the network traffic must follow – called a policy.

The simplest firewalls have a packet filter, which specifies the ports that are allowed to be used – often port 80 and 443 for `HTTP` and `HTTPS` respectively. Added to this, is stateful firewalls, which resembles a NAT to some extent.

The capabilities of a firewall is not as clearly defined as a NAT. A firewall could have any number of rules that might check destination, port, content and any other parameter, to determine, if the packet should be blocked or let through.

A normal setup for a firewall is to only allow `HTTP` and `HTTPS`, as the administrators might find other traffic to be suspicious in a working environment[13]. This kind of firewall, which limits the kinds of protocols to use, will also be looked into in section 3.4.

1.4 Partial Conclusion

The way the Internet has developed, it has limited the access to devices from the outside world. We have a few obstacles that has to be traversed, which we later will look at ways to circumvent. There is a little hope, that it will become easier in the future with IPv6, but it is still uncertain.

The ongoing sections will focus on how to make cameras, printers and other LAN-connected devices available from the Internet.

2 Proposition

From the background in the previous section, we have set forth a few requirements. We want to access devices connected to a local network. These devices are not currently accessible from the Internet because of either a firewall or NAT.

Looking at the obstacles we have, the quickest way to enable this kind of access, would be to open a connection from each device to a central server. This server would then work as a proxy to facilitate the user.

This solution has some big downsides. First of all, someone would have to pay for the central server. Secondly, The server would also have to scale linearly with the amount of devices, as every device would take some of the resources from the server – even idle devices would require a file descriptor as a minimum.

My proposition to solve this, is to install a small piece of software on all of the employees's computers in a company or on all the computers in a household. This will enable a type of peer-to-peer (P2P) network, that will take care of interconnecting machines. Only one or two of these devices would have to keep an open connection to the Internet. Here, we have already reduced the amount of connections, if there are more than one device on the local network.

Aside from only keeping one open connection from each LAN, we could also detect, if a computer is directly connected to the Internet and therefore able to receive inbound connections. This computer could be linked into a network of helpers, that could off-load the central server.

This way, if we assume that a certain percentage of the users have open access to the Internet, we can bend the curve for the amount of resources the central server needs. If for example $\frac{1}{50}$ users add one of these helpers to the system, and we assume, that each of them can handle more open connections than the frequency of occurance (for example 75 each). This way we will bend the curve and have a constant capacity of the central server, but the users are adding resources at a self-sustaining rate.

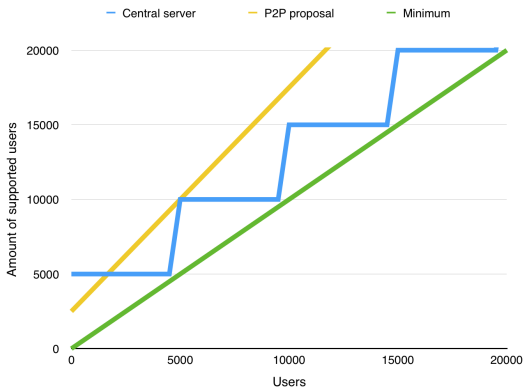

Figure 2: Illustration of relationship between resources and users

On figure 2 is shown an example of the relationship between users and capacity of the system.

Green Line
: The minimal amount before the users would see outage from the server.

Blue Line
: The central server, which gets upgraded in bumps of 5000 users.

Yellow Line
: The combination of a central server with a constant capacity of 2500 users with the addition of helping users.

The network itself is not meant to have any functionality to the end-user. The network is used by another application on the users system, that wants to utilize the network. The network sets up the infrastructure, enabling interconnectivity, but otherwise generate no other traffic.

When interconnection has been accomplished, the software can also be used to distribute load away from the central servers. It will be at the core of the project, to minimize and distribute workload, to enable rapid scaling of the system.

2.1 Peer-to-peer networks

P2P networks are a form of network, which utilizes common users, to provide a service, that is normally enabled by servers. It

has become widespread in file-sharing software, which includes BitTorrent, LimeWire etc.[4] But software like Skype, has also adopted the P2P structure to minimize server-load[5].

In contrast to BitTorrent and file-sharing networks, which emphasize on decentralization and resiliance to governments or lawenforcement trying to stop it[6]. The purpose of this network is to distribute workload. This network also has a central control point much like Skype. The central server has to validate users and put central control on the otherwise decentralized infrastructure.

The software is imagined to be used by a company that provides a service and let the company have control of the network. This gives the network some special characteristics that limit certain aspects, but also enable us to take advantage of others.

File sharing is a perfect case for P2P, as the object being transferred is completely stand-alone and it can be validated by checksums, to make sure nobody has altered the content. If somebody sent you a file and the checksum does not match, the software will just find another peer to download the data from.

Before Popcorn Time and P2P streaming became mainstream, it did not either have a lot of requirements in perspective of constant availability[17]. The peers did not need an uptime compared to a server. The difference that a single peer can make, is insignificant in contrast to the hundreds or thousands of peers for a specific file.

2.2 Performance

For a service on the Internet, we can imagine a centralized server, and all its clients as a connected graph.

If the system works by having a single meet-up point, having a central server is the most optimal solution, in respect to minimizing connections, as each client makes one connection each. Although it is not the most optimal for sharing resources.

This is the way a lot of services work, but it does not scale well, as the central server is handling *all* of the traffic, and it has a finite

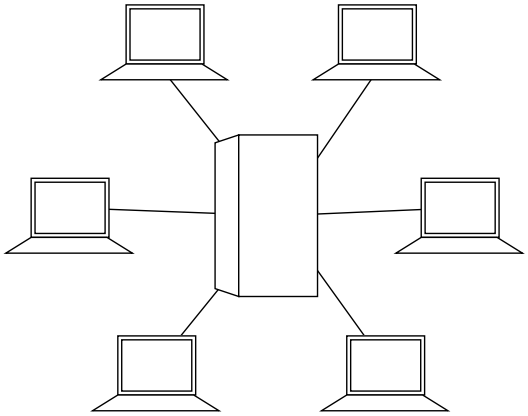

Figure 3: Traditional server with multiple clients

amount of resources. For such a system, we can assume that each connection adds to the load of the system. Meaning, that the more connections, the more load is put onto the system. This means, as the service becomes more popular, the system runs out of resources and the owner has to upgrade the server to scale with the traffic.

Looking at a P2P network, as the popularity rises, so does the capacity of the system. If each user on average adds more resources to the system, than it takes, the system will theoretically scale equal to the demand.

Looking at the downside of P2P network, the global amount of connections and the global load on a P2P network will always be equal to or higher, than the design used for a central network. The design in figure 3 is the most effective in terms of minimizing the amount of connections, as there is exactly n nodes and $n-1$ connections. It's possible for the P2P network to construct just as good patterns, but it would still have the overhead of constructing this network.

As illustrated by figure 4 which has $n-1$ connections to connect all node in a loop, it would still require some kind of redundancy (the dotted lines) in case a node disconnects. The P2P network will practically always add more connections to the global load to maintain the structure of the network.

Although the global load of the P2P network is higher, than using a central server, it adds load in a different way. Instead of

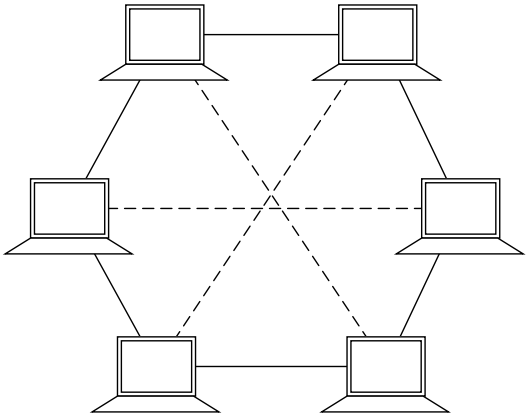

Figure 4: Figure 3 reinvisioned as P2P

having a single point, it is distributed out between a large amount of peers. From the standpoint of the peer, there is almost no measurable difference between having one constant connection to a central server, or a hand full of connections to other peers in respect to power consumption, or workload on the computer.

The philosophy of P2P networks can be described with an analogy. If a group of a million people is given one dollar from one person, that individual has a huge loss, as he paid for everything. If we reverse it, and say that every person in the group pays *two* dollars to a specific individual, each participant, has not suffered any significant loss, but the total amount of transferred money is now doubled.

The point that can be drawn from this, is that the global load on a P2P system might be higher, but the perceived penalty from each individual is neglectable.

2.3 Overview

The end result of the project is to connect a computer to a device, where the device is behind a firewall, NAT or another common restriction. This has to be done without configuring the local network by means only available to a system administrator.

The devices we want to connect to, could be a printer, surveillance camera or another LAN-connected device. Normally, we cannot expect to install software directly onto the device, as these are often loaded with a firmware that only the manufacturer has access to.

The solution will be to proxy the connection through a local computer, that is in contact with the outside world.

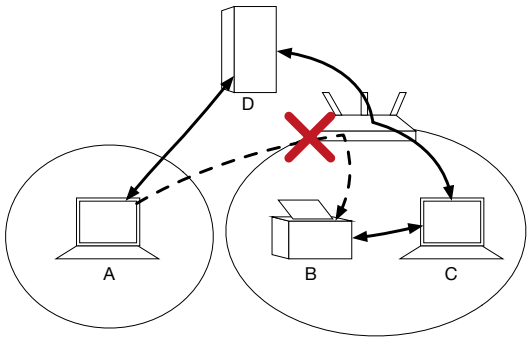

Figure 5: Getting access to a device through a proxy

It is expected, that the computer that should be working as proxy has basic access to the Internet. With this, we can have a server with open access to the Internet, that the proxy computer is in contact with. This server can work as middle man and receive requests on behalf of the proxy machine, and resend the information. This server will most likely have to proxy the data between the proxy on the local network and the computer that requested the connection.

2.4 Partial Conclusion

From the requirements we have seen in the introduction, we have put up a plan to fulfill these requirements while also enabling scalability. As part of the peer-to-peer system, we enable scalability by not having a linear relationship between the central server's load and the amount of users on the network. The peer-to-peer network will add to the global load of the system, but as perceived from a users standpoint, the added load will be almost imperceptible.

3 Design

We have decided to solve the problem, by constructing a peer-to-peer (P2P) network. Because of this, we have some design decisions at hand. To have the groundwork before implementing the solution, we have

to get an overview of the system and describe how to fulfill the requirements.

3.1 Overview

We will contruct a few examples that we can consider during the development of the system. Ideally, these should have been actual customers of the potential system, but we'll make up some realistic scenarios.

3.1.1 Scenario A

We have a scenario of a company with a surveiliance camera and two computers on a local network. The company uses a normal consumer-grade router to connect everything to the Internet. The router uses a NAT, which is making none of the devices available from the Internet.

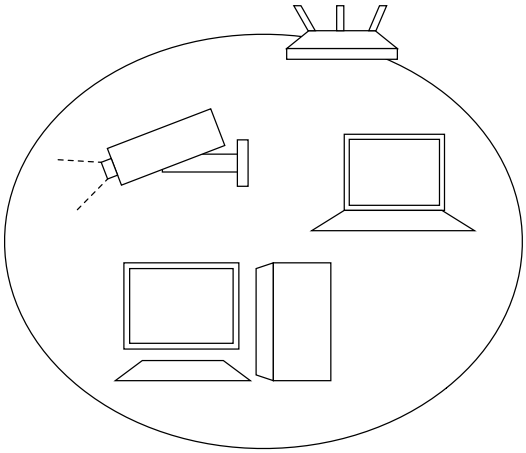

Figure 6: Scenario A – A camera and two computers

3.1.2 Scenario B

We have a scenario which contains Scenario A, but adds an extra company with similarities. One of the companies has a server, which is used for various things. On this server, the P2P software is installed.

3.1.3 Scenario C

Abstracting away the internal construction of each company, we have several companies. Some of these companies have installed the software on servers that can function as redundancy. We will look into these servers in section 3.8.3.

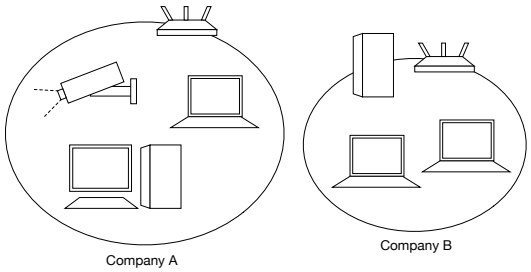

Figure 7: Scenario B – A server and two laptops

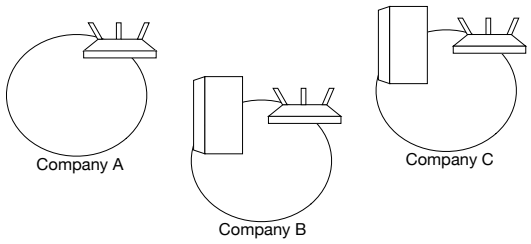

Figure 8: Scenario C – Three companies, two servers

3.2 Locating Devices

One of the key problems, is for the software running on a user's computer to detect, if it is qualified to proxy data to a local devices.

Every network-connected device has its own unique serial number, called the physical address or Media Access Control address (MAC-address). The MAC-address for a computer's IP-address can be found by broadcasting an Address Resolution Protocol (ARP) packet on the local network[15].

3.2.1 ARP-scanning

One way would be to check, if the wanted device is accessible every time we connect or disconnect from a network. This could work, by noting the MAC-address of the wanted device, and do an ARP-scan to see if its MAC-address comes up, and on which IP address. This kind of scan would be effective on smaller networks, but it does not scale well, if the subnet becomes larger. And the firewall might detect it as port-scanning and block the communication.

When one or more devices have been located, it would announce it to the P2P network, that computer A sees printer X and camera Y. If multiple computers are on the same network, it would mean that the net-

work is announced $n \cdot m$ times; where n is the number of computers and m is the number of devices.

3.2.2 Identifier

Assuming that multiple devices are often accessible from the same location, it might be favorable to create a identifier for the location of several devices instead of registering accessibility for each of them. Would it be possible to construct a unique location-identifier for each location a device is accessible? And what makes a location unique?

The first uniqueness of a network location must be the router that joins the devices and computers together. All routers have a unique MAC address which identifies them. But what if a router has a segmented network, where some users are located on one subnet and others on another subnet? Could they see the same router with the same MAC address? In that case, we can add the subnet mask and the IP prefix to the identifier.

We now have the router's MAC address, its subnet mask and the IP prefix. All this information should be possible to collect from a user's computer without stressing the network with more than one ARP call for the router's MAC-address. These three components are hashed together, to form a unified identifier, that can be used for lookups. The hashing gives a slight obfuscation, but it also unifies the length of identifier, which might help, when implementing the database.I will discuss the issue of obfuscation in section 4.1.

When the user's computer finds out it is at the location of a device, it announces to the network, that computer A is at location X. A computer could have multiple accesses to the Internet, through WiFi, ethernet or VPN. All these would generate each its own identifier, where only some of them might give access to some devices. This means, that a computer might be registered at multiple locations at once.

3.3 End-devices

The term end-device will be used to specify any kind of device, that the system will allow connections to. This would normally be printers, security cameras or some other Internet-of-things device, possibly another computer.

It is not part of this project to identify or use the end-devices, but only to recognize if they are present on the network and allow other software to use them.

The end-devices are not supposed to be the peers themselves. Although nothing in the system will actively try to stop any device from becoming an end-device.

The end-devices are registered outside of the P2P network and are announced on the P2P network.

3.3.1 Identifier

Each device has to be uniquely identifiable on the network. An identifier is constructed from the device's MAC-address and a hash of the SSL certificate, if the device has SSL. These two values are hashed into the end-device's identifier.

The identifier together with other information about the device is announced on the network and can be retreived when a user wants to connect to the device.

3.4 Obstacles

The reason we cannot always connect directly into an end-device or a computer on a network, is not always the same. Some companies uses firewalls to limit the protocols or other uses of the network. The limit is often based on allowing specific ports – for example 80 and 443 for `HTTP` and `HTTPS`. But could also go deeper and limit use by deep packet-inspection to block connections using port 80 that is not `HTTP` [13].

In small companies without an IT-department, it might just be a Network Address Translation (NAT) that is in the way between the open internet and the user.

3.4.1 Hole Punching

NATs work from the assumption, that a local computer has initiated an outbound connection before an inbound connection can occur. This works well for web browsing and most tasks on the Internet, as the user asks a webserver for content and gets an answer back.

The NAT keeps track of inbound and outbound connections. When a datapacket is being send out, the NAT inserts an entry in a table saying "computer A send a packet from port 4365, this is mapped to the external port 9854". When a connection comes in, it searches the table to find a local computer, that would be expecting a returning packet. If the incoming packet was sent to 9854, it would be redirected to computer A, if no entry was found, the packet is discarded. This effectively means, that routers with NAT does not allow computers on the internal network to act as servers, as they cannot receive any inbound packets.

The quickest way to get around this limitation, would be to constantly be in contact with a server and have it proxy any request. All computers wanting to communicate, would have to be in contact with the same server (see figure 9).

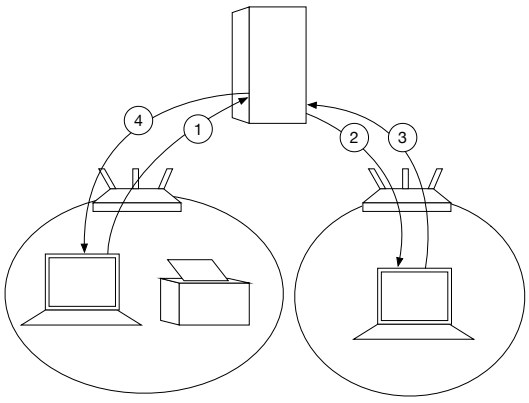

Figure 9: Illustration of proxying through a NAT

This works, but it is ineffective, as a server on the outside will have to continuously send data back and forth. A solution to this, is a technique called hole-punching. It tricks the NAT, by making an entry in its internal table from computer A to server A, but in reality, it's computer B that utilizes the now open route into the network. See figure 10 for an illustration. The steps are:

- Computer A and computer B connects to a server
- The server informs the computers about the port and IP of the opposite computer
- The computers A and B connects to each other through the IP and port they got from the server

Depending on the implementation of the NAT, computer A will be able to use the open port in the opposite NAT, that computer B got opened to contact the server and vice versa.

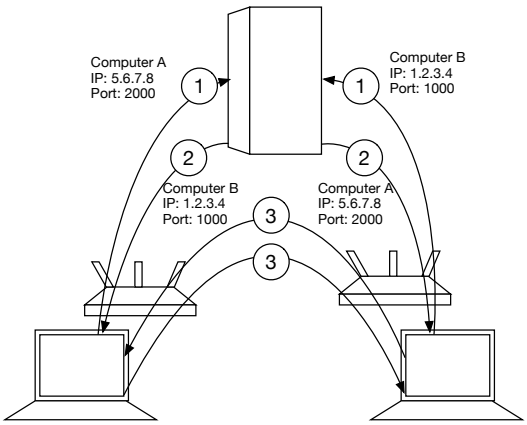

Figure 10: Illustration of hole-punching

The reason the computers cannot just contact each other is because the technique has to be coordinated. They have to know about each other and the NAT will, depending on implementation, disallow an inbound connection, if the time between the outbound and inbound packet is too far off.

The scheme becomes a bit unpredictable depending on the implementation of the NAT. Some NATs use not just the local sending IP and port for a table entry, but also the receiver's IP. Meaning, that computer B cannot use the NAT record that was used for the server. There are some ways to circumvent this too, but they are not always successful.

One could for example expect the NAT to assign external ports sequentially, meaning that if the external port was 4000 for the

server, the NAT record for the next connection, must be close to this value – depending on other traffic on the network being assigned a port. Then the two sides, knowing the IP and port that was used for the server, could send a burst of UDP packets to ports from the previous port and a few ports up, to see if it can guess the correct port. This might cause the opposite effect, if a firewall detects it as portscanning and blacklists the senders IP.

3.4.2 Firewall

Different implementations of firewall can have very different ways of working. For simple cases, the above example with hole-punching might work, as it tricks the firewall into thinking the inbound connection from computer A or B was expected.

For other cases, it might block specific port and protocols. For those cases, there is not a simple work-around that will always work.

For the common case, where only `HTTP` on port 80 is allowed, the P2P software could try to shift protocol and port, to circumvent this restriction.

A more creative example would be to utilize a ICMP or DNS tunnel, which encapsulates data into a ICMP or DNS packet to a server on the other side of the firewall.

3.4.3 UPnP

In some setups – particularly home networks – with a NAT, we can find a local UPnP controller that will help us traverse the NAT by appointing us a specific external port upon request.

This works by indentifying the UPnP controller on the network and send it a request through an XML-based protocol. This request could be: The IP 10.0.0.10 wants to have the external port 2222 forwarded to the local port 22. The controller checks the port is unused and otherwise allowed and either replies with an accept or denial of the request.

For this to work, the P2P network has to be flexible on the account of which port to use. It is not always possible to get the requested port in case of restrictions or multiple users on the same network.

3.5 Tunnels

The main purpose of this project, is to solve the usecase of connecting to devices behind firewalls. To solve this, the use of tunneling is found to be a solution.

Tunneling is the concept of carrying one datachannel through another. A common example of this, is to use SSH's tunnel feature to access a website which is only available from the inside of a network. The SSH tunnel also provides encryption between the user and the tunnel-machine. For this project, encryption will not be offered, but will be recommended to use on the inner datachannel.

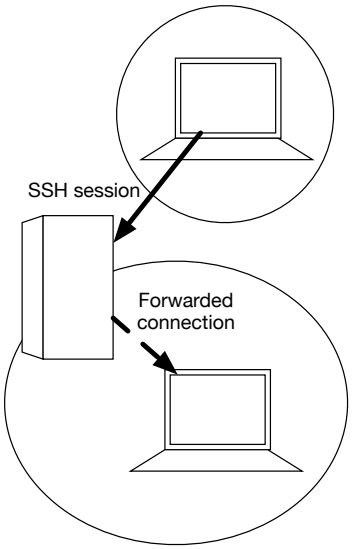

Figure 11: SSH Tunneling

By having a computer on the same network as the end-device, we can use this computer to resend and thereby tunnel data into the local network.

To connect to an end-device, we first have to find out where it is. The third-party software requesting to make the tunnel has to know the identifier of the end-device. A description of the identifier can be found in section 3.3 above.

The computer that wants to set up the tunnel, sends out a query on the P2P network, to find information on the device. If successful, information about computers

that shares location with the end-device is returned.

From the list, one or more computers are contacted to initiate a connection. When the connection has been established, the computers on each end verify, that they are from the same company or household and is otherwise valid computers of the network before opening the tunnel. We will get into details about verification in section 3.7.

The computer that is being used for the tunnel, contacts the end-device on behalf of the computer that want to contact the end-device.When the end-device responds, the tunnel-computer does nothing, other than resending the data it receives.

3.6 Trust

When building a computer system, that involves third-parties, which you have no control over, you have to think about who to trust. Can you ever trust someone on the Internet you do not know? The answer is a very short, clear and loud: *NO!*

Especially when building P2P systems, where users can become part of the infrastructure, it is absolutely vital, to be prepared for people trying to disrupt the system.

Therefore; this system is built up in the mindset, that only the people running the central servers can be trusted – as you presumably pay the owners to run the network and install their software. Every other computer on the network has to be treated as hostile and expected to try to gain unauthorized access or make the network unstable.

On this bombshell of socialphobia, we have to make the system create some kind of trust. How do we create trust and stability from utter chaos?

The following sections will focus on encryption and digital signatures, which will be used to validate any data or any computer, that we will contact. On top of this, section 3.9 will also set up some rules of contacting other computers, which will prepare us to handle faulty, unstable and malicious computers.

3.7 Domains, Keys and Identification

To make some trust in the system, I've chosen to give every computer its own public and private key. This will be used to sign every transaction of information on the system. This will help to keep the integrity of data, when computers share information about each other.

The central server of the system, which we will call the *operator*, has to sign every computer's public key. The operator will only sign the public key for a computer, that can verify to be a member of one of the companies or organizations, which pays for the service. The verification is not part of this project, as this would be handled by Active Directory, OAuth or some other authentication service.

The public key will also be used as an ID-number by hashing it. Whenever a computer contacts another computer, it will include its ID-number. If the receiving computer does not recognize it, a request will be sent to get its public key and verify, that the public key has been signed by the operator's private key.

By using signatures, we can verify any node's public key without asking the operator. If we did not have the signature, we would have to verify public keys by contacting the operator all the time.

As the system is expected to be running as a cloud-service, the central servers will have to differentiate between computers from seperate companies and organizations. This is handled by creating a 32-bit number for each organizations and prefixing every computer's ID-number of that organization. This identifier will be called the *domain*.

	Domain	Public Hash
Company A	df e4 73 65	98 c9 84 82 14 b3 b5 80 a4 05 45 …
	df e4 73 65	b5 80 a4 05 45 7a f1 fc 80 d0 9f …
	df e4 73 65	05 11 ec 33 7a f1 fc 80 d0 53 79 …
Company B	51 9a 6b cd	74 66 ed 33 7a f1 fc 80 d0 53 79 …
	51 9a 6b cd	72 11 54 82 14 b3 b5 80 a4 05 45 …
	51 9a 6b cd	97 92 a4 05 45 7a f1 fc 80 d0 9f …

Figure 12: The domain associates nodes from the same company

The domain also ensures, that tunnel-nodes are only exposing devices to nodes that are part of the same domain.

The domain does not technically need to have a reference to a geographical location in the real world, but it could be used to limit latencies. We will get into the advantages in section 3.10 and possible risks of this in section 4.3.8.

3.8 Nodes

For the further development of the system, we need a term to describe a computer with the P2P software installed. We will call this a *node*, like a node in graph theory. To simplify the structure of the system, every computer is a node. The users, central servers and so on, are all nodes running the same software. The operator-node will of course have different responsibilities than a user-node, but the basis is the same.

A node can have different features enabled. All of these features are chosen by the node itself. This does allow for nodes to falsely advertise they are more capable, than actually permitted. It will be up to the design of the system to detect this and ignore a node like this. I have considered to let a central server decide which nodes can enable certain features, but in reality, all criteria for such a selection can be forged anyway (ie. performance, availability and so on).

I will get into other kinds of forgery and attack vectors in the section 4.3.

The following subsections will describe different kinds of nodes. It is possible for a node to be a user-node, tunnel-node, super-node and operator-node at the same time, as non of the modes overlap.

3.8.1 User

All nodes start off as a user-node. It has no special abilities and is mostly silent on the network. It will have to keep itself alive by reregistering at the operator-node from time to time. If the time expires, other nodes will no longer recognize the node, and are allowed to remove information about the node locally. If removed from the system, the affected node will simply resend its public key to the operator-node.

The specific time should be set from a judgement of how quick the network should adjust to nodes leaving and joining the system. Setting the time to 30 days, seems like a level, where inactive nodes are removed at an acceptable rate, and not too often to burden the system.

The user-node is the only node, that does not keep contact with other nodes regularly. The other types will periodically check its IP address and check it's the one known to the operator-nodes.

3.8.2 Tunnel

A tunnel-node is working as a middle-man to connect a user-node to an end-device. The tunnel-node will automatically find out, that it can reach an end-device and will announce itself as a tunnel-node through the P2P network. The tunnel-node will possibly have to send packets to one or mode super-nodes to keep an open connection for incoming requests. See section 3.4 for more details.

An active tunnel is expected to be running for no more than a few minutes. Although nothing currently enforces this nor limits the amount of data through the tunnel.

3.8.3 Super

Super-nodes are any regular user-node, that has open access to the internet. The soon-to-be super-node finds out that it has this capability and requests to join the network by asking the operator-node. The operator-node assigns it to a group of existing super-nodes.

This idea of a super-node is similar to how Skype operates [5].

A super-node is used to off-load the workload from the operator-node. There are two kinds of workload. Look-ups on the node-database (we will go into how this database is constructed in 3.10) and proxying a tunnel. For look-ups, the super-node is assigned a specific partition of the look-up table. This partition is shared with other

super-nodes and has to be kept synchronized between super-nodes for that partition.

Some times, a super-node has to act as proxy between a tunnel-node and a user-node. This happens if both ends are behind a NAT or firewall, that the system cannot traverse.

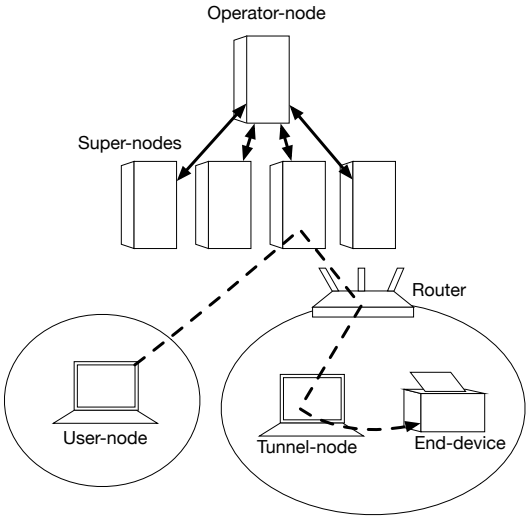

Figure 13: Utilize super-nodes to reduce workload on operator-node

The partition that a super-node is appointed, could be chosen from its domain. By doing this, a company that has a super-node running, will have a more direct benefit, by utilizing its own hardware. It's also more likely, that a company has its super-node in close proximity to its users, and thereby lowering the latency. We will look at some of the risks of doing this in section 4.3.8.

3.8.4 Operator

The operator-node is the central point of this P2P network. It is controlling who is allowed on the system and how to form the super-node structure. Its function is mainly to have an initial contact for every node. Either for the first time a node starts, or if the network experiences a crash.

The operator-node is the only trustable element of the network. The central control helps ensure the systems integrity.

3.9 Protocol

For communicating between nodes, we will have to design a protocol. The protocol will include the abstract structure of messages, the data they carry and how to check for authenticity.

This section will only handle the ideological contents and purposes of the protocol. In section 5.5 we construct the datapacket. Later, we will also look at ways to serialize it and in the section 5.5.4, we will look into how we transmit the data.

3.9.1 Messages

Nodes communicate using a messaging system, this system ensures authenticity and to some degree, security.

The messages will work as a type of RPC in the system. The messages will mostly be used to submit information and retreive it from another node. Messages will also be used to call special features, like setting up a tunnel.

The nodes on the system will have to exchange information about each other; who is super-node, who is at this location, what end-devices are available and so on.

These messages are all short and have a clear request-and-response paradigm.

The messages will be fairly strict in the ways they can be used. The messages will contain a specific command number followed by a context (like a parameter to a function). It should not be possible to execute arbitrary code on a remote node, only asking a specific question and receiving a response.

As a policy, if a node receives a message which is malformed or in some other way unusable, it will always default to discarding the message. It will not tell the requesting node, that it even received the message.

If we hypothesis, that we tell the requesting node, that an error occured. What would the affected nodes do about it? If it's a bug which causes this, they would not be able to fix it. If the receiving node is maliciously replying that a message failed, we will just waste our time. The best way to handle it, is the same way, as the receiving

node never got the message and move on to another node. We will get into this in section 3.9.4.

3.9.2 Statelessness

To build a basis for a robust system, I have chosen to make the messaging system completely stateless. Nothing is remembered between two messages, as this could open up to a variety of exploits and generally make the system more complex than needed.

The node's primary task is to wait for a message to come in. When a message is received, it handles whatever action that must be taken and returns back to square one. The only state a node has, is knowing who it is.

Because of this loop of receiving and handling messages, it is important that no message has any significant waiting time. For example receiving a message and then waiting for a response to come in is disallowed. This would lead to ways of blocking a node, by making it wait on purpose.

By making the protocol stateless, we also open for the possibility of running the actions in parallel. This might be beneficial for the operator, if the CPU cannot handle all incoming messages on one thread.

The nodes use a local database to cache information about other nodes, but this is not considered part of the node's state, as it will be recreated automatically, if lost.

For operator-nodes and super-nodes, they will have to remember a state of which nodes to be in contact with. This state is not recoverable, in case of a failure on the node. In section 3.9.4, we will look at how to work around failing nodes.

The only node to have an exact state, that will cause serious problems if lost, is the operator-node. This node will have to keep track of all super-nodes. In case the operator-node fails, the super-node system will have to reinitialized.

3.9.3 Encryption and Signing

As every node is equiped with a cryptographic key pair, we can use this in the messaging system, to make sure who we are communicating with. Just because they have a key pair, does not make them trustable, but it removes any anonymity, and makes it possible for the operators to remove them from the system – and in the real world, take legal action. The lack of anonymity is in respect to the P2P network. Only the operator has access to data correlating IDs to people in real life.

Whenever a node receives a message, the first thing to do is to check its signature. If the signature is invalid, the whole message is discarded. The message could have a failed signature either because of a data corruption while transmitting or if the message has been altered. In either case, there is no way to use the message.

It would also be possible to encrypt the contents of the message, but any middleman will always be able to see that node X sent a message to node Y and read the checksum.

3.9.4 Fault Tolerance and Redundancy

A P2P network in inherently exposed to nodes going offline and online at any given moment. The thing that makes the network function in the real world, is to allow for such randomness and always be ready to move on and recover.

The stateless nature of the messaging system enable us to quickly recover from a node that goes offline. Because of the statelessness, any node is equally qualified to handle a message. This of course has the prerequsite, that a tunnel-node is at the correct location and a super-node is within the same partition as the failing node.

Retreiving information When contacting a super-node to retreive some information, it is possible that this node is offline or too busy. To prepare for a possible error, we send out the same requst to multiple super-nodes.

We talk about the number of redundant requests with a redundancy of degree K. This will of course double, triple or otherwise multiply the total load on the system, compared to only sending one request. But

there are two aspects to this. First reason is to reduce the risk of waiting for timeout for something that is never coming. The second reason is to prepare for a super-node which does not have the newest information or is intentionally holding back information. We will get back to the second argument in section 4.3.3.

Redundant super-nodes When a super-node joins the network, it gets a partition of the P2P network to handle and one or more redundancy nodes to cooperate with. The operator-node is used for redundancy if there are not enough super-nodes. It will be announced to user-nodes, that the super-node is available.

User-nodes will have a list of super-nodes and operator-nodes in their local databases. When the user-node starts up, it will start from the top of the list and cycle through the super-nodes or operator-nodes, until it finds K nodes, that respond. The found nodes will be reused until one or more fails.

In case all operator-nodes or super-nodes in the list does not respond, the action will fail and a request for a new list will be sent (this is only done once).

3.10 Distributed Hash Table

I've meantioned a node can announce something. But how is this achieved? We need a system, where nodes can put in information and let other nodes access it.

P2P systems like BitTorrent use something called a Distributed Hash Table (DHT)[22]. Just like a normal hash map or dictionary, it stores a key and value pair. The distribution of it, is what makes it work in a P2P network. By strategically dividing the table onto a range of nodes, and assume that the keys used to store information are uniformly random, we can effectively distribute the data and the workload.

On top of distributing the work across servers, we have the opportunity to use our super-nodes as workforce to drive the DHT.

The DHT works by requesting a specific hash-key to retrieve a value. All hashes are prefixed with the domain of where it comes from. This enable a bit of consistency and motivates a company to spawn a super-node to get direct use of it.

The domain will be used to partition the DHT and optimize super-nodes to serve the same domain's user-nodes. Even though this opens for a sybil attack, read more in section 4.3.3.

We will let the super-nodes handle all of the traffic for the DHT. If a query falls through on the super-nodes, it might be considered to allow the operator-node to be asked in some cases.

Each entry in the database contains the actual data to store, but also a timestamp, a signed checksum and who signed it. In case of a node's public key, it is signed by an operator-node. All other entries will be signed by the specific node, that it concerns. Having this restriction makes it theoretically impossible to forge data in the system – for example changing the IP-address a node announces it has. Changing a value, will cause a mismatch for the checksum and no node will accept it.

The restriction, that only a node itself can upload a entry for itself, also solves concurrency issues, as there will only be one writer.

For more specific details in regards to implementation, see the section 5.4.

3.10.1 Partitions

One way to make the DHT, would be to replicate the entire table across all super-nodes. This high redundancy level would make it very stable in terms of super-nodes going offline. Although it would suffer from difficulity to scale properly, as every super-node would require an increasing amount of storage space and an increasing amount of bandwidth just to keep up to date.

Kademlia A popular way, this issue has been solved in the past, is the Kademlia DHT[14]. This kind of DHT was considered, but it did not fit the requirements well enough.

Kademlia is simply a single table of 128-bit hashes that map to a value. Every node on the system has an ID of 128-bits, just like the hashes of data being stored. Looking up a value, is done by asking the node

you know, which has the ID that is closest[1] to the hash. The asked node returns the node it knows, which again has the ID closest to the hash. This process repeats until the hash and value is found or there are no more nodes to ask.

This system has the benefit of great scalability and it is completely decentralized.

The downside comes, when you want to update or remove a value. This is effectively impossible, as there is no way to find every single node, that has a value stored. It also assumes, that all nodes are readily accessible to the Internet, which is the exact opposite assumption of this project.

Kademlia's strong suit really is file-sharing, as files are completely static and it is not as important how fast a node can query the DHT – the recursion takes time. As the download time often out-weighs the query.

Partitions The strategy that I will use, spun off of the Kademlia system, and shares some of the same perspectives.

I will use several tables to seperate the different datastructures.

Each table will be distributed across the super-nodes on the P2P network. I will split them in *partitions* based on their associated domain. Each super-node will have two attributes stating the lowest and highest domain ID it will serve. It is not a requirement, that the super-node's own domain ID is within this partition.

The operator-node starts the network by having responsibility over all partitions. When a super-node joins the network, the operator-node will give it a partition to keep track of and send it all the values the partition currently has. The super-node will also get a list of other super-nodes to keep up-to-date with.

Depending on the degree of redundancy we require, we will start off, by having multiple super-nodes clone the entire table. When enough super-nodes has joined, the operator-node will begin to subdivide the super-nodes, preferably in a way that require the least amount of data to be exchanged. At first, the user-nodes will not be informed by the change of super-nodes. They will learn this on their own. When a super-node gets its key-space reduced because of new super-nodes joining, it will stop responding to requests out of its key-space. This way, we will make the user-nodes find out, that something has changed and avoid distributing old data.

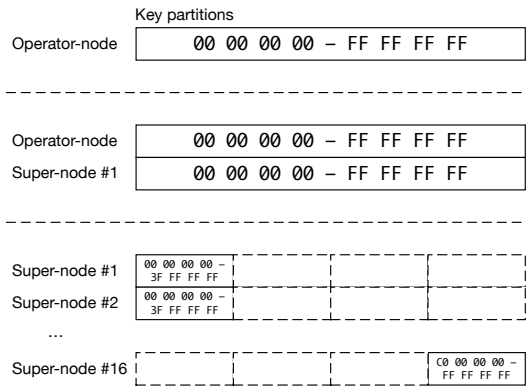

Figure 14: The key-space is distributed for redundancy and divided to improve performance as the network grows

When user-nodes come online, they ask for a new list of super-nodes, therefore it's only the older user-nodes that has to be considered. These older user-nodes will keep using the same super-nodes until one of them goes offline. Then, as described above, the user-node will find a new super-node if possible, otherwise it will ask for an updated list from the operator-node.

3.10.2 Data loss

Data that is static on the system, like a node's public key, will always be kept on the operator-node as back-up. Data that is dynamic and changes frequently throughout the usage of the system, will only be updated on the super-nodes. This is both to avoid overloading the operator-node, but also because the information that changes, are often easy to reproduce – hence we can handle it carelessly.

In case of a super-node failure where all redundancy is lost, it will be up to the operator-node to redivide partitions of the key-space to the remaining super-nodes. The data that was lost, is not directly re-

[1] Kademlia uses `XOR` as the distance function[14, p. 4, XOR metric]

coverable and the user-nodes will experience an outage. Although this is also the reason why tunnel-nodes and super-nodes reannounce themselves frequently, to reconstruct after an outage and to keep it updated.

3.11 Logic and scenarios

With all the information from the previous sections, we will set up some scenarios to show how a node will act in different situations.

3.11.1 Initialization

Every time a node turns on, it will check it's local storage for a number of saved properties.

All node must have a public and private key, for signing communication. If this key pair is not found on disk, the node will generate a new key pair and send the public key to the central servers through `HTTPS`. The central server will sign it, to later be able to verify, that the key is allowed on the system.

When the key pair is generated and added by the operator to the DHT, the node can begin to query and store information on the DHT.

In a case where a node has a malformed key pair or it has been removed, it will register as if it never have been on the network before. The old key pair will expire and get removed from the system, when the node stops using it.

3.11.2 Retreiving information

With the new ability to query the DHT, we want to update our list of operator-nodes and super-nodes.

The first time our node contacts a super-node (or any other node), the receiving node will not have the public key of our node. The receiving node will discard the sent message and reply with a `GetNode` message.

Our node will reply the message with our public key, which is already signed by the operator. The super-node will then check the signature and use it for messages it receives later.

We will not get any verification, that the public key has been received, but we can retry our first request.

3.11.3 Failing super-nodes

If the node has a list of super-nodes cached, it will continue using those, until they stop working.

In a scenario where a super-node fails, we cycle our list of super-nodes one-by-one. If we run out of super-nodes, we retreive a new list and cycle through it.

3.12 Partial Conclusion

I have made an overview of what scenarios I will try to solve. Part of the solution is to be able to identify what network the user is on, and if it matches an end-device of interest.

I have looked at the obstacles I would expect in a normal user-case and how to circumvent them.

We have gone through the subject of trust and how to introduce some kind of control in a decentralized system using cryptography.

I have introduced the concept of a node and how different types of nodes will be used in the system.

I have outlined the structure of communication and how it will be handled, that nodes should be stateless.

I have chosen to use a Distributed Hash Table (DHT) to spread information on the system. The DHT will mostly work as a cache on each node to decrease latencies and unnecessary load on the system. It will also allow nodes to share information inbetween each other.

4 Security

As this system is distributed and highly accesible from the Internet, it is worth considering the risks of the system. The more software gets distributed, the more it will be exposed to getting hacked in some way or another. As I anticipate attacks on the

system, it might be wise to prepare for it and go through the considerations.

4.1 Privacy

Privacy is not strictly a security issue. You can have a perfectly secure system, which is controlled by a company that doesn't care for the user's privacy by selling information to third-parties – thereby breaking the privacy.

On the other side, it's hard to secure people's privacy without locking away the information.

In this P2P network, we have the challenges that any user can become super-node and store information about any user. It's important for this information to not contain anything, that can be abused.

The nodes are all uniquely identified by their domain and public key. If one were to correlate the domain to a specific company, it would be possible to monitor when users begin to come online as the tunnel-nodes begin to be announced. I can't determine if this broad information about a company is a significant risk. It is most likely not, as one could find this information without the P2P network.

But what about a household? It would be possible for a thief to monitor when a family leaves the house. But is this really a secret? I would not be surprised, if it was every day between 8am and 4pm, where nobody was in the house. Apart from the fact, that the thief would have to determine which house is using a specific domain and public keys.

If one were to correlate a specific public key to a person, it would be possible for a super-node to monitor when this user goes to work, takes home and so on. At this level, you would be able to determine alot of information about a persons whereabouts. But remember, that the super-nodes only get notified when the user is becoming a tunnel-node, a super-node or if the user is querying the specific super-node. It would still be hard to determine which public key to monitor.

In section 3.2, I mention the MAC-address for routers are hashed, which has the side-effect of obfuscation. At the same time, end-devices have their MAC-address written directly in the information. A MAC-address is bound to the specific network card in each computer. But is this sensitive information? In itself, I wouldn't say it's personal information, especially for routers, printers and cameras. It becomes personal, if it is a wireless device which the user carries around. In this system it is not. It is only used for devices which are stationary and most likely connected by ethernet.

The system doesn't directly give away private information, but the information can theoretically be correlated to be abused. The most important obstacle for abusing this information is to determine the domain and public key of interest. The common user will be hidden in the crowd.

4.2 Encryption

To avoid a range of impersonation attacks and privacy issues, all communication has to be signed, and in some cases encrypted, mostly to verify the sender and receiver. All information exchanged between two nodes, will be in the form of a small messages (see section 3.9.1). These messages will be encrypted with the receiver's public key and signed by the sender's private key; thereby only enabling the correct receiver to read the contents and verify the sender.

The operator-node gives access to retreive public keys and other information about nodes. All public keys on the system *has* to be signed by an operator-node to be valid. All nodes will individually verify the validity of a node, when receiving a message, by retreiving the public key from the sender and verify its signature from the operator-node. The initial operator-node certificate will either be programmed into the software or retreivable through a `HTTPS` connection – putting trust on the operating-system's certificate authority validation.

4.3 Attack Vectors

In case of any design or implementation decision, it must never be allowed to accept

a security flaw because "it might not happen" or "it seems unlikely". And to this extend, security has to come through encryption and signing, making the only accepted security flaw be of obtaining a private key. Although denial-of-service attacks are not strictly security issues, they will be discussed in this section.

4.3.1 Gaining Access

If the software is programmed perfectly, the only way to gain access, is to obtain the private key of the target node, for example by gaining file-access to the node's machine. Having a copy of a node's private and public key, will enable the attacker to impersonate the node without any restrictions. Gaining access to a single user's node will have a minimal impact on the system as a whole. The attacker might be able to decrypt any data between the target node and other nodes.

4.3.2 Abusing an operator-key

It might be of higher interest to get a copy of an operator key. This will enable the attacker to add unlimited nodes under any domain.

With the operator's key, it will be trivial to dismantle the infrastructure between super-nodes. The attacker would be able to create a high amount of super-nodes on the system. Then by disconnecting all other super-nodes, gain complete control of the information, and control any tunneling.

4.3.3 Take over the super-nodes

Would it be possible to take down the network from spawning malicious super-nodes?

Without access to an operator key, it will still be possible to disrupt the network. Depending on the amount of existing super-nodes, it could require a large amount of nodes to perform a sybil attack. A sybil attack is based on systems, where normal users can gain part of the operation of the service. Some networks have voting systems. If any user can create an unlimited amount of nodes, this can be used to vote in favor of oneself or to distort the data the super-nodes are storing[9].

Let us define the basics before digging into the math. Nodes always query K super-nodes, let's define $K = 2$. See appendix A for the chances of hitting a malicious super-node.

At system startup, it is only the operator-node on the super-node network. The first super-node joins as redundancy for the whole key-space. When nodes query the system, they will currently ask both the operator-node and the single super-node, as they are the only two on the system. So far, the system cannot be compromised by the super-node.

To simplify the following calculations, we use the super-nodes as redundancy for the whole key-space.

As the number of super-nodes rises, we stop using the operator-node. Now, we do not have any specific super-node that is trustable.

I propose a solution, that will mitigate the issue of rogue super-nodes, under a few assumptions:

- There is at least one truthful super-node
- Super-nodes either send back outdated data or no data
- The requesting node can detect a modified response from the signature
- The time of which to retreive the data is not limited

If we have an example of 10 super-nodes, where only one is truthful. How would this in worst-case play out?

The user-node has a list of the super-nodes in randomized order, and starts querying super-nodes one at a time. If we get no answer or receive an entry that is too old, we cotinue to the next super-node on the list. We might get an entry that is completely valid and not outdated yet, but the super-node knows, that a newer version exist. In this case, we still accept it as an answer. Otherwise we continue until all super-nodes have been tried.

To speed up the process, the user-node could start by querying two super-nodes asynchronously. If no valid result is found, then double the querying by sending out 4 messages. The doubling continues until the list is exhausted or a result has been found. By doubling the number of requests, we try to get the result faster, but with the expense of stressing the super-nodes. Although this will only be temporary, as the user-node will remember which super-nodes were giving a valid result.

This example would require up to 10 messages being sent out, but at a maximum of 3 timeout-periods. To speed up even further, it would be possible to send out the messages at a shorter interval than the timeout, but still wait for an answer for the full duration of a time-out.

4.3.4 Chain-reaction denial of service

In a scenario, where a large amount of user-nodes are connected to the system, and they are evenly distributed across the super-nodes. If a super-node goes offline, all the user-nodes it was in contact with, will have to find a new super-node. If all user-nodes use a list of super-nodes, that is sorted in some way, they will start contacting the same super-node, as it's the next on their list. This new super-node will then have its workload doubled in a short time, as the user-nodes detect the first super-node has gone offline. If this super-node cannot hold the pressure of suddenly handling all user-nodes from the previous super-node, it will also crash and cause a snowball effect.

This chain-reaction will keep going until all super-nodes has crashed, or a super-node is reached, that can carry the pressure from all previous super-nodes.

To avoid this scenario, all user-nodes will have to randomize the order of operator-nodes and super-nodes. This will (with a good random number generator) statistically cause all of the affected nodes to distribute evenly across the redundant super-nodes. If all super-nodes are under heavy load, the chain-reaction might still cause more super-nodes to crash, but that case is not possible to handle, without increasing the general capacity of the super-node group.

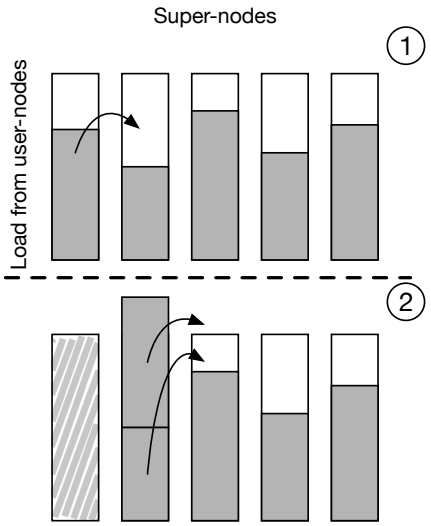

Figure 15: Snowball effect of super-nodes crashing

4.3.5 Monitoring rogue super-nodes

Can we determine if a super-node has gone rogue?

The ideal solution would be for user-nodes to vote down the rogue super-node, but this would open up for sybil attacks [2][9].

Instead, it would be possible from the operator's side, to create a spy-node on any domain and query a super-node to see if the response is correct. This way, we would get an independent judgement of a super-node. This could result in isolating a super-node – for example by making it inaccessible, but without telling it so. It would be possible for a determined attacker to recognize the IP of the spy-node, but that part of the cat-and-mouse game is left out of this project.

4.3.6 Denial of service from tunnel-nodes

Would it be possible to announce false tunnel-nodes and effectively make user-

[2]Sybil attacks are caused by users setting up multi identities to overrule the behaviour of the system[9]

nodes unable to find a working tunnel-node?

Yes, to a certain degree. Contrary to super-nodes, that are shared across all users, the tunnel-nodes are only shared internally for a domain. This means, that it is only possible to spawn tunnel-nodes for the same organization as the attacker is a part of.

This means, that for a user to take down the system this way, it could only be an inside job. This of course is still possible, but I've allowed for such an attack, as I found it to be an internal affair of the customer and it will not affect any other domains of the system.

4.3.7 Denial of service by flooding the super-node databases

For the super-nodes, there has to be set a practical limit for how many entries are allowed in the database. Not because it is technically demanding for the system, but because the super-nodes are treated as borrowed hardware and should not be burdened, if under attack. Another part is isolating and monitoring possible errors or attacks.

If we say the limit for a super-node is 10,000 entries in the database, would somebody be able to fill it and thereby block or purge another domain's entries? The answers is currently: Yes, in some cases. If a super-node gets filled up, it could do a garbage-collection of entries that has expired. This will hopefully make room for the first time this issue occurs. If a node is deliberately trying to fill up a super-node, this solution will not continue to work.

If the limit of 10,000 entries is shared across all domains on the super-node, this will be easy to exploit. A possible workaround would be to have a limit which is individual to every domain. This way, it would only disrupt the specific domain, if somebody is trying to clog the network.

There are a few ways to store information on the system. By adding nodes, end-devices, or announcing tunnel-nodes for a end-device. All of which are prefixed with the domain of the issuer. This will make it clear what domain is faulty or under attack and isolate it.

4.3.8 Targeting a specific domain

As described earlier, the domain can be used to optimize queries by letting a domain host super-nodes that will serve its own users. This opens to an attack, where the attacker gets a domain that is close to a target domain. The attacker spawns a super-node and gets a partition which covers both its own domain, but also the target domain. Then the attacker is free to disrupt exactly the user-nodes from the target domain, as they will begin to use the super-node.

Currently, the only way to detect this, would be to use a spy-node controlled by the operator. This spy-node could probe the attacker's super-node as if the spy was from the target domain.

4.4 Threat Level

When making the assessment of the system, we will take on the role as an attacker and look at what goal we would want to accomplish.

The likelihood assessment is from a perspective of a network with around 5,000 to 10,000 users spread between hundreds of companies. Some of these companies might have business intelligence of interest to a hacker.

The most exposed part of the system would be the use of a common Rust-library for HTTP, this would enable broad attacks, if a flaw were to be found. Apart from this, it will likely require a high focus to get anything useful out of an attack. The low-consequence attacks might not require a high level of skills, but just the right amount of resources. The realistic attacker would likely be in the high-skill, high-focus, as any of the other groups will have a very little benefit from the attacks.

In appendix C, I will go through the risk assessment of the five most likely attacks.

Figure 16: The expected attacker of the system and what they could accomplish

4.5 Logic and scenarios

With all the information from the previous sections, we will set up some scenarios to show how a node will act in different situations.

4.6 Initialization

Every time a node turns on, it will check it's local storage for a number of saved properties.

All node must have a public and private key, for signing communication. If this key pair is not found on disk, the node will generate a new key pair and send the public key to the central servers through `HTTPS`. The central server will sign it, to later be able to verify, that the key is allowed on the system.

When the key pair is generated and added by the operator to the DHT, the node can begin to query and store information on the DHT.

In a case where a node has a malformed key pair or it has been removed, it will register as if it never have been on the network before. The old key pair will expire and get removed from the system, when the node stops using it.

4.7 Retreiving information

With the new ability to query the DHT, we want to update our list of operator-nodes and super-nodes.

The first time our node contacts a super-node (or any other node), the receiving node will not have the public key of our node. The receiving node will discard the sent message and reply with a `GetNode` message.

Our node will reply the message with our public key, which is already signed by the operator. The super-node will then check the signature and use it for messages it receives later.

We will not get any verification, that the public key has been received, but we can retry our first request.

4.8 Failing super-nodes

If the node has a list of super-nodes cached, it will continue using those, until they stop working.

In a scenario where a super-node fails, we cycle our list of super-nodes one-by-one. If we run out of super-nodes, we retreive a new list and cycle through it.

4.9 Partial Conclusion

I've outlined a large range of security concerns. Most are related to disrupting the system and cannot always be mitigated. In case of attacks, it will be priority to isolate it to a specific domain. In which case, it will be up to the operator to manually exclude the customer from the system or other enforcements to stabilize the system.

5 Implementation

This section will solely focus on the actual implementation of the P2P software. Any design decision has already been made in section 3.

5.1 Language and Libraries

For the project, we want a object-oriented language, preferably cross-platform and allows for code with a low memory-foorprint

and low CPU-consumption. As we are also writing a network protocol, it will be preferred to have strict control of data types, as they will have to be serialized and deserialized in an efficient and correct manner.

5.1.1 Rust

I've chosen to use Rust as the language for the project. Before starting the project, I had never written a single line of Rust.

Describing Rust in short term, it is a general-purpose, object-oriented programming language, which is inspired by functional programming languages, but is written as an imperative language[8, Appendix: Influences]. One of the key features of the language, is the static borrow-checker, which analyzes the code at compile-time and ensures memory safety[7, Introduction].

For dependency handling, the system `Cargo` is used, which also automatically handles compiling all the source code through `rustc`[3].

Rust is still a fairly immature language, but has a very active community, which updates every 6 weeks [20].

5.1.2 Serialization

We have to communicate over a data-connection. Rust does not have a direct way to transform a native datastructure into a serialized form. Rust has a very limited set of frameworks and modules in the community, but I found a library called bincode (not to be confused with b*e*ncode, which BitTorrent uses)[1].

It has a fairly simple format, although for this exact project, it adds a little more overhead, than needed. It reads out the native datastructure from Rust into binary data. But in front of all arrays, it prefixes the length in the format of a 64-bit integer. This is of cource perfect if you intend to store large amounts of data, but for arrays of less than 64 bits of data, the overhead is larger than the data. For the hashes of 32 bytes, this will add 25% extra data as overhead and 200% for the domain of 4 bytes.

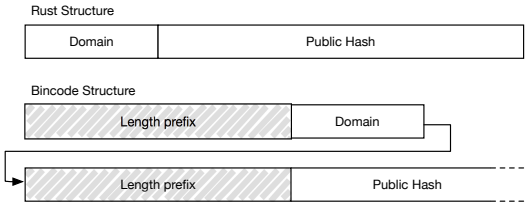

Figure 17: Illustrating overhead of ***`bincode`***

5.1.3 Database

For the DHT, we need a library that can work as a database. We could implement a simple hash-map in memory, but it requires a lot of extra work if we want to be able to save, load, and search for specific entries.

I found SQLite to fit the case perfectly. It might not be optimized for performance in the same way as enterprise-level SQL servers, but SQLite is easy to install as standalone and does not require a lot of setup to work with.

On GitHub, a user named "jgallagher" has made a wrapper, which allows for easier interaction with the underlying SQLite implementation directly from Rust[10].

5.1.4 Encryption

The encryption on the system has yet to be implemented. The software has been programmed exactly like it would with encrytion working. But all "encryption" is handled by placeholder-functions. Some dummy functions just return true when checking, while I validate that the transmitted data is being encrypted through a simple `XOR`-encryption.

The final solution would likely utilize OpenSSL, PGP or another asymmetric encryption method.

Because I have not chosen a specific method, there have not been any consideration as to how a node's key could be revoked. It could hypothetically be given in the specific way to encrypt. If I would have to design it myself, I would persumably add a type of list to the DHT of revoked keys. This would grow in size over time, but there are ways to solve these issues.

For the same reason, I have not added an expiration time for the keys. There is no

way to determine how it would be handled without a specific method to reference.

5.1.5 Hashing

The software will need a library to do cryptographic hashing on several occasions. The most important one is how we identify each node by a hash of its public key. The second occasion is when messages will be signed, by encrypting a hash digest.

As the encryption in not implemented in the system, neither is the hashing of public keys nor signing of messages.

The expected implementation would use SHA-256. Which will be reflected by the byte-size of the appropriate datastructures in the next sections.

Collisions for the SHA-256 are highly unlikely to happen by accident in the size of this system. If it should happen to a user-node, it will only require one of the involved two node to reinstall, and it will most likely never happen again. If one were to fabricate a public and private key that fits exactly to the hash of an operator-node, they would not be able to use it for anything, as any node will reject its signature. If they were to replicate the private key while finding a hash collision, they would of course have access to everything an operator-node can do.

5.2 Initialization

When the node starts up, it check the SQLite database. If it's not present, it will be created, as we will need it to store information about the operator-node.

Then it will check if its key-pair is working, by loading it to memory and doing a simple integrity check. If not, it will generate a new pair and register it at the operator-node through `HTTPS`.

To ensure we have the newest list of super-nodes and operator-nodes, a request is sent off for both.

The node detects if it is being started as an operator and enables the states for structuring the super-node network.

At last the node enters the message loop and waits for incoming connections.

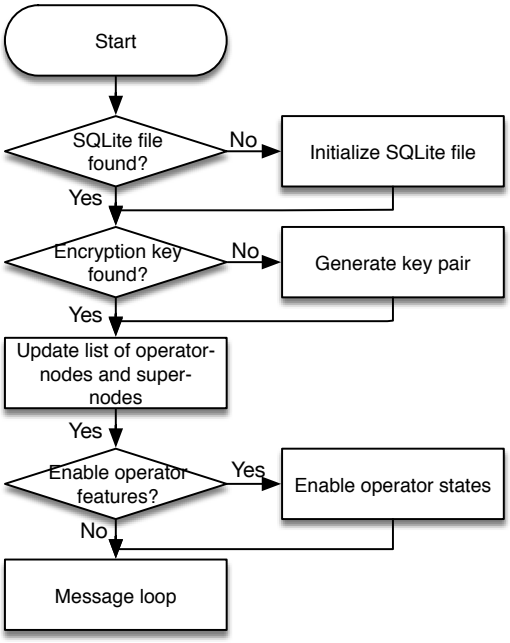

Figure 18: The flow of initializing a node

5.3 Data Structures

To build the DHT, we need a set of datastructures, which will represent the different nodes on the network. We need the structures, to share public keys, inform other nodes about our IP and our listening port for messages.

All hashes on the system are calculated using SHA-256 and are stored as byte-arrays.

5.3.1 Node

This is the static datastructure for a node, it's the one that will have to be updated every 30 days (defined in section 3.8.1). It'll always be stored on the operator-node.

Domain	4	bytes
Public key hash	32	bytes
Public key	n	bytes
Timestamp of signature	64-bit	int
Signee's domain	4	bytes
Signee's public key hash	32	bytes
Signee's signature	32	bytes
Total	$112 + n$	bytes

The size of the public key is denoted by n because the encryption is not yet implemented. Therefore it has not been decided

what the key size will be.

The reason for storing the signee's domain and public key is to prepare for multiple operator-nodes or for the time when an operator-node's key has to be changed.

It's important to note, that it's only the operator-node's private key that is valid for signing the datastructure. Other signatures will technically be possible to make, but no node should accept it.

5.3.2 NodeInfo

This is the dynamic part of the node-structure, this will never be seen by the operator-node. It'll be stored on the super-nodes so user-nodes can find tunnel-nodes. The NodeInfo data will only be removed from the DHT when the coresponding node is removed or when the update interval expires.

Domain	4	bytes
Public key hash	32	bytes
List of IPv4 addresses	4	bytes $\cdot\ n$
List of IPv6 addresses	16	bytes $\cdot\ m$
Listening port	16-bit	int
Timestamp of creation	64-bit	int
Location ID	32	bytes
Node's signature	32	bytes
Total	$78 + 4n + 16m$	bytes

The list of IPv4 and IPv6 addresses are followed by n and m. These denote the number of addresses being stored. This is because we may find multiple interfaces which each has one or more addresses – for example one from each of ethernet, WiFi and VPN.

It's a known limitation, that there is no way to determine which IP-address that the listening port is refering to. Just like there is no way to determine if the IP-address is for a VPN connection.

In this structure, there are no "signee" fields, as it has to be the node itself, that signs it – hence the two fields on the top are used.

5.3.3 SuperNode

A super-node will have the two structures above to define the basic parts of the node. As an extension, the following structure will be added to the DHT by the operator-node. Its purpose is to create a relation between which super-nodes are assigned to a specific partition of the DHT's key-space.

Domain	4	bytes
Public key hash	32	bytes
Partition from	4	bytes
Partition to	4	bytes
Timestamp of creation	64-bit	int
Signee's domain	4	bytes
Signee's public key hash	32	bytes
Signee's signature	32	bytes
Total	120	bytes

As with the node-structure, it's only the operator-node that is valid for signing this structure. This makes sure, that no third-party changes the super-node network.

5.3.4 OperatorNode

The operator-node's structure is the only one that does not contain any additional information. It's just a proof that it is accepted as operator.

Domain	4	bytes
Public key hash	32	bytes
Timestamp of creation	64-bit	int
Signee's domain	4	bytes
Signee's public key hash	32	bytes
Signee's signature	32	bytes
Total	112	bytes

The signature is a special on this one. It is signed by itself, as it is the operator. But the structure still allows for one operator-node to sign another operator. This allows to mitigate from one key pair to the next, or adding an operator-node on the fly.

5.3.5 End-device

This datastructure is also statically saved on the operator-node to make sure it does not disappear.

One would expect there to be a TunnelNode structure, but as user-nodes can find the tunnel-nodes through the location they

are reporting to be at, it becomes redundant.

Domain	4	bytes
MAC-address	6	bytes
Hash of SSL	32	bytes
Location ID	32	bytes
Timestamp of creation	64-bit	int
Signee's domain	4	bytes
Signee's public key hash	32	bytes
Signee's signature	32	bytes
Total	150	bytes

For a user-node to connect to an end-device, it will have to query the super-nodes to find the end-devices available to its domain. The end-devices are handed over to the third-party software, which is using the P2P network.

When the third-party software want to use the end-device, it will find the tunnel-nodes which are at the right location through the super-nodes.

5.4 Distributed Hash Table

The DHT's purpose is to have a database, which is not fully contained on a central server, but is rather distributed amoung users of the network. When node stores information into its DHT, it has to check the entry to be digitally signed by the correct node and otherwise is correctly structured data.

Because of the asymmetric encryption scheme, it is possible to independently validate, that a specific node has signed an entry in the DHT.

The DHT is implemented through a combination of the messaging system and a local SQLite database. The practical part of the messaging with be touched in the section 5.5.

SQLite is an SQL implementation, which is embedded into end-user software and runs as a local SQL server and client. A database from SQLite is contained within a single file on the user's system, which makes it easier to handle. The way SQLite works is in contrast to traditional SQL servers, which is often a central server, which other services connect into.

The tables of the database are build to contain the datastructures described in section 5.3.

When a node wants to store something on the DHT, it has to go through two steps:

1. Save the data into the node's local DHT
2. Send the same data to the appropriate super-nodes

When the data is stored at the super-nodes, it will save it to its local SQLite database and it will instantly become available for other nodes to find the information.

The local copy is not strictly necessary for all situations, but it is a matter of consistency and not handling unimportant corner-cases.

5.5 Messages

For the implementation, the messages are transferred through single UDP packets and messages are serialized directly into binary data in the packet. The final solution should be able to adapt to a possible firewall, which might ban UDP packets. This could be resolved by automatically switching between UDP and TCP for the transport layer and using `HTTP` or a more efficient protocol, whenever needed or permitted.

5.5.1 Packet

Messages are serialized from structures native to Rust when transmitted, and deserialized on the receiving end. The current implementation serializes the message to binary data using the bincode library (see section 5.1.2 about libraries), but this may change in the future. In the beginning of the project, the data was serialized to JSON, but the code allowed to switch to bincode fairly easily.

5.5.2 Command

A command structure is embedded into the messages, that nodes send. The command is the essential part of the message, which triggers an action.

The messaging system can only be used for a limited set of command types. All of the commands are handled independently and no session is carried between messages. Although there is no session, it is still expected, that certain messages, like `GetNode`, are answered appropriately by returning a `Node` message. Still keeping in mind, that we have to handle situations where a message is not delivered or is not answered.

The command type is followed by a payload. This payload is used as a form of parameter or context, for the command to use. Some command types, like `GetOperatorNodes`, has an empty payload, while the returned `OperatorNodes` contains a list of operator-nodes in its payload.

An overview and explanation for each of the command-types can be found in appendix B.

5.5.3 Structure

Messages consists of the following structure:

Receiver Domain	4	bytes
Receiver Public Key Hash	32	bytes
Sender Domain	4	bytes
Sender Public Key Hash	32	bytes
Command	n	byte
Time Stamp	64-bit	int
Signed Checksum	32	bytes
Total	$112 + n$	bytes

The payload is denoted by n, because the size of the payload is dynamically increasing as needed. Although the full package cannot be larger than what UDP and IPv4/IPv6 allows. We will look at this limit in section 5.5.4 below.

The checksum is calculated by taking all the other fields and concatenate them to a list of bytes. Then, we use SHA256 on the bytes, to calculate the hash digest. This digest is signed using the *sender's private key*. This will make sure, that any message can be validated against the *sender's public key* to make sure of its origin.

The time stamp is a safety measure, to make replay attacks harder. Although it does not solve the problem completely – a third-party could still resend the message within the validity period. If the payload is encrypted, there might not be any reason to do a replay attack, other than disrupting a node using another person's identity.

The payload should be encrypted in the future using each node's private key. The current implementation uses a simple `XOR`-encryption with a fixed key, to verify the payload is being encrypted.

5.5.4 Carrier

To transfer the messages, I will use UDP packets, as these share similarities with the messages. They are connectionless, relatively small and doesn't provide any guareentees in respect to delivery and response.

UDP packets has just 4 fields of a total of 8 bytes: Source port, destination port, length and checksum. These fields are followed by the actual data of the packet. The length field is 2 bytes long, which allows to allocate 65536 bytes. Some of the bytes are already allocated to the headers, which limits the packet to carry 65507 bytes of data in total.

The end-devices we expect to interface with, are presumed to work over a TCP connection. Therefore, we need a way to setup such a connection. I do this, by sending a packet through UDP, which makes the software open a port for TCP.

5.6 End-devices

End-devices are signed with the operator-node's private key and inserted into the operator-node's SQLite database from another software running on the operator-node's computer.

At a specific interval, or at manual operation, the changes to the end-devices are sent out to the super-nodes handling the appropriate partitions. The user-nodes will not immediatly get the new end-devices, but the third-party software, which is using the P2P network, can trigger an update.

5.6.1 Tunnels

The end-devices are expected to communicate through TCP and possibly `HTTP`. It's

not important what the protocol is on the application layer, as the data should just be retransmitted. Although the application layer should use SSL or equivilant to avoid the tunnel-node reading the data or possibly manipulate it in transit.

When a node want to create a tunnel, it sends a `GetTunnelNodes` message to the super-nodes, which has their location at the specific ID, which the end-device has.

From this list of tunnel-nodes, we have three possible situations for each node:

Tunnel-node is also a super-node The tunnel-node has an open port to receive inbound requests on and can receive the inbound tunnel-request directly.

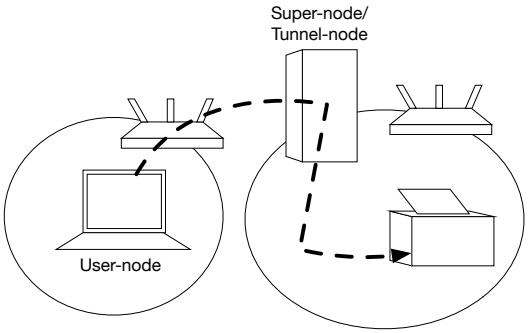

Figure 19: Tunneling through a super-node directly

In this situation, we send the node a `TunnelRequest` directly and initiate the tunnel.

Tunnel-node with access to port forwarding Some nodes might be behind a router that has proven to work with hole-punching or UPnP forwarding. A super-node can be requested to assist with hole-punching, but otherwise, the tunnel-node will open a UPnP port.

The procedure for this, is to get in contact with the tunnel-node through one its connected super-nodes. We send a `ForwardedTunnelRequest` to the super-node, which redirects it to the tunnel-node. The tunnel-node opens a port using UPnP or hole-punching and answers the super-node with a `ForwardedTunnelOpened`, which is sent back to the user-node.

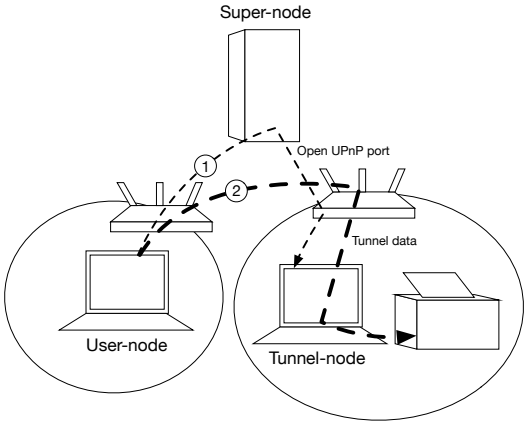

Figure 20: Opening UPnP port and tunnel directly

With the IP and port included in the `ForwardedTunnelOpened`, the user-node initiates the tunnel.

No inbound connections The tunnel-node is behind a firewall and cannot receive inbound connections. A super-node has to assist in proxying the tunnel.

This is the least desirable option, and should be avoided, if any of the above methods are possible. This works, by initiating a connection from the super-node through the tunnel-node to the end-device. The super-node sends the user-node a `TunnelOpened` with its own IP-address and an open port. Then the third-party software on the user-node connects directly to the super-node, as if it was the end-device.

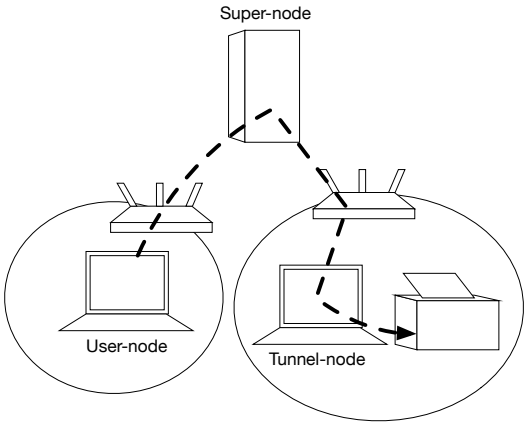

Figure 21: Tunneling through both super-node and tunnel-node

Only the first situation has been implemented and tested, as the other two options require specialized network hardware or equivilant software, which I did not have access to.

It can be noted, that all of the normal node-communication is handled through UDP with the messaging system. But for the tunneling, we send a message over UDP to "upgrade" to open a TCP connection.

With the current state of Rust and its I/O implementations, I was forced to open two threads for each tunnel. Meaning, that one thread waited on a socket-read from the user-node to the end-device, and vice versa on another thread. The implementation that I wanted to make, would have a single thread handle all tunnels, by waiting on multiple reads at once. Whenever data would come in on a socket, it would trigger an equivilant write, and return to waiting for data from any of the open tunnels.

5.7 Partial Conclusion

I have looked more specifically at how I will implement the system. I've chosen to program and learn Rust for the development of the project. It has been chosen for its modern take on a programming language and supposedly good performance.

Although Rust does not have a broad variety of libraries, I've still managed to find a library for SQLite to save the DHT and a serialization system called `bincode`.

Some parts of the system did not get implemented, especially encryption and signing. They have been implemented as dummy-functions, which verifies their use.

I've outlined the datastructures, which will be used to store nodes and their special properties. The datastructures have specified fields, which will be used to sign the structure and valdiate its integrity.

6 Performance

For any application, and especially network applications, we have to look at the stability of the design and implementation.

It's important to have a design, that has been thought out to handle error situations, in a manner where the system can recover quickly. This has been a major focus of this project, because of the indeterministic nature of P2P networks. If we were in the lucky situation, that all nodes had a proven 100% uptime, and nobody was trying to disrupt the network, we could have made some highly optimized networks.

Although the design of the software, is the keystone to make the software function, it is just as important to look at the implementation.

Even if the software is designed in a perfect manner, the implementation will always differ from the ideal implementation. In this section, we will look at the stability and performance of the software and reflect on the future possibilities.

6.1 Stability

Stability of software is hard to measure. There is not really a final day of coding, where you declare "all features has been implemented and the system is stable".

From a theoretical standpoint, one could create unit-tests to check error-handling and features for every possible situation. In reality, this becomes a tedious task, which is limited to central points of the system or particularly critical parts. From a practical standpoint, it is impossible to try out and check every conceivable situation, that the software might be exposed to.

Stability often comes from trial-and-error. First you run the software in a simulated setup and see if you can make it behave wrongly. You correct any mistakes and test it again. When the software *seems* to be stable, you move it to production and do nothing, but wait for the users to complain.

The cycle of errors and fixing them will possibly keep going forever. But one would hope, that the amount of complains will fall, as the software "matures".

6.1.1 Handling errors

By design of the software, it has been made easier to handle errors. If an error occours

while processing or receiving a message, the fallback solution is to discard the message and return to waiting for a new message. And the other nodes, that are waiting for a callback from a discarded message, will expect this behavior and assume the node is either down or unable to process the message – either way, there is nothing to do about it – and already be on the way to find an alternative node.

This design makes the ideal implementation resiliant to random software crashes and if the errors keep occouring, other nodes will be ready to replace it. Other nodes will one by one stop asking it, as it seems to be error-prone or offline. But this does not stop the failing node from asking others, if that part of the software is unaffected.

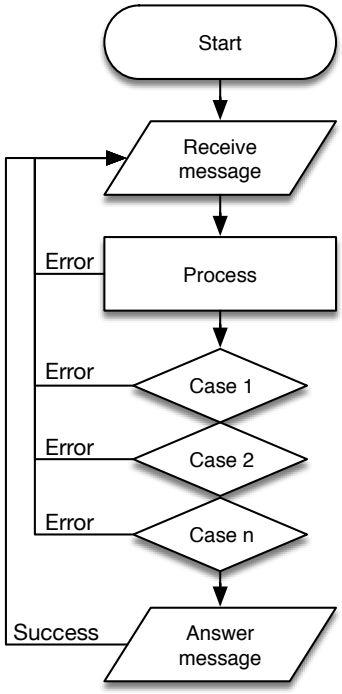

Figure 22: Flow of messages. Any error discards the message.

6.1.2 Testing node

It can become challenging to test the stability of software by fuzzying random data into a single node[18]. Any message that comes into the software has to go through several steps before it hits my code. The results of the test would be the stability of the I/O implementation and deserialization of third-party libraries.

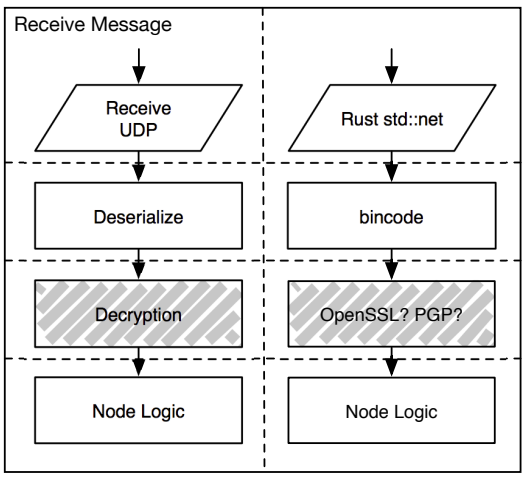

Figure 23: The three layers of third-party code before the node's logic

Although it would be possible to carefully contruct messages, which traverses the third-party libraries and carries out tasks that intentionally fails in my part of the code. The only constructive result we would get from this, would be bugs that keep correct requests from being carried out.

Rust doesn't have a concept of exceptions, rather `Option` types, `Result` types and if everything breaks, a `panic`. The first two are "unwrapped" safely through a pattern-match to graciously handle possible errors. The `panic` is only used in situations where there is no other possiblity, than to terminate the whole thread.

I know that the message handling part of my code doesn't have any `panic` statements. Therefore, it will always be able to reset to receiving a new message. If a `panic` were to happen, it would be out of my hands to catch it, and possible an error within Rust itself.

6.1.3 Testing network

What would be more interesting, would be to test if we could deconstruct the P2P network by making key points unstable. This would take a step back from making a specific function inside the software to crash, and focus on the network as a whole.

I have been going through several situations that could be challenging for the system in the section 4.3. Although one could try to set up a scenario with hundreds of nodes spread out over complicated network configurations, a test at this scale does not seem to be within reach of this project.

6.2 Benchmark

The software is coded in Rust, which gains efficiency from being compiled, and from a philosophy of having a strict control on memory, threads and other system resources. But how efficient is it really? I want to put it to a test to see what the limits or throughput would be in highly stressed situations.

For the tests, the code has been compiled for release, which enables a range of optimizations and strips out some debugging routines.To check if this is true, I ran the first test in debug and release mode. In release mode, I saw a 26% percent increase in throughput and execution time is cut by 68%.

6.2.1 Test computers

The main test computer, which will be the user-node in the tests, is a MacBook Pro from late 2013. The computer does not have an ethernet port, but will be connected through WiFi to an IEEE 802.11ac compliant router.

The secondary test computer, which will be used as the operator-node in most of the tests, is a Mac mini from late 2012. It will be connected to the same router as the MacBook with access to the same subnet.

	MacBook	**Mac mini**
CPU Ven.	Intel	Intel
CPU Name	i5-4258U	i5-3210M
CPU Gen.	Fourth	Third
CPU Freq.	2.4Ghz	2.5Ghz
CPU Cores	2	2
CPU HT[3]	Yes	Yes
RAM	8GB	4GB

Granted, that none of these computers are the ideal environment for running a server, it will still show what speeds, that we would expect as a minimum. On a sidenote, these results, would probably be in line of what I expect the end-user's computer is capable of.

[3]Intel Hyper Threading

6.2.2 Message throughput on loopback

I want to see what kind of throughput in messages per second I can get from sheer computing power, without factoring in bandwidth limitations of ethernet. The expected outcome is to completely utilize both cores of the processor.

For the test, I set up an operator-node and a user-node on the MacBook. The user-node is set up to send `GetNode` requests as fast as possible in one thread and receive responses from the operator-node in another thread. These threads count outgoing and incoming messages. The test runs for 10 seconds.

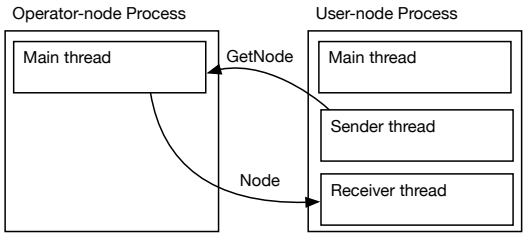

Figure 24: Setup of the two node processes

The amount of incoming messages on the user-node should be the maximum throughput, as it is expected to saturate the thread of the operator-node, which will be forced to drop some of the messages.

The first time I did the tests, I set up a node to send out 1.000.000 requests and make another node respond to them. By measuring how many messages that came back, I expected to find the maximum throughput, as the superfluous messages to be discarded. This way of testing had the oversight, that a slow connection resulted in better throughput. Because the loopback interface is too fast, it can send the 1.000.000 requests within a few seconds. The receiver cannot process them any faster, on a slow or fast connection, therefore most were just dropped.

I ran the test a range of times and record three tests. While testing, I ensured that the CPU utilization came close to 100% for both processes – verifying my previous assumption.

Returned Messages
76,412
71,400
68,564
72,125 Avg.

Even though the CPU utilization is close to 100%, I want to rule out, that the SQL part of the `GetNode` command is waiting for IO. Therefore, I try the same test, but with a custom command type called `Ping`, which responds with `Reping`.

Running the new test, results in the sending node having "114%" CPU utilization, where OS X measures 100% as one core. This increase is likely because the operator-node can reply more often, which occupies more time on the receiving thread.

Returned Messages
124,745
116,286
122,833
121,288 Avg.

This change gives a increase in throughput of 68%, which must be due to skipping a SQL query for every message. Although it's hard to say, if it's IO on the SSD that adds this penalty, as it's the same query every time, the data is likely cached by the OS in memory. Looking at the system's IO activity, shows almost no activity in both test cases.

6.2.3 Message throughput on LAN

This test is exactly the same as the previous one, but instead of running through a loopback-interface on the MacBook, I'm sending requests from the MacBook to the Mac mini. Monitoring the amount of data being transferred on the previous test, doesn't show more than 3MB/s to 8MB/s. This means that the use of LAN instead of the loopback interface, might not change much in the test.

Returned Messages
74,169
74,580
73,395
74,048 Avg.

This result was surprisingly close to the loopback-test. It even got almost % more messages through; although this might just be a coincedence. My theory is still, that this test is bound by the requests going out to SQLite.

Looking at our `Ping` again, we will see the performance without the SQLite calls.

Returned Messages
108,391
107,031
106,748
107,390 Avg.

The test shoulds a slight decrease in throughput compared to the loopback-test. Through LAN, we still got 89% of the same throughput from before. It is still above the 100,000 messages per second, which I did not expect. This might show we are hitting a limit of the LAN connection, compared to the previous test that gave almost the same results.

6.2.4 Tunnel throughput

I want to see, if there is a penalty to using a tunnel in matter of throughput. This is an important part of the system, as end-devices like a printer, would be bound by the transferspeed, if the print-stream is not getting received at the speed of pages coming out of the printer.

To test the speed, I created a file of 1GB containing random data.

```
dd if=/dev/urandom of=testfile \
bs=1048576 count=1024
```

I send this file through `scp` from the MacBook to the Mac mini to test the throughput of a plain connection.

Seconds	MB/s
95.12	10.77
94.80	10.80
94.99	10.78
94.97	10.78

The transferspeeds are surprisingly low. I was expecting speeds of at least 40MB/s and up to 80MB/s from previous experience with the same two computers. I have not been able to find the bottleneck, so I will have to continue the tests with these speeds as reference point.

Running it through a tunnel, we get the following results:

Seconds	MB/s
94.78	10.80
101.39	10.09
92.44	11.08
109.54	10.56

The throughput is almost the same as the previous test. There is a bit more spread in the last two transfers, but I would say it is nothing of importance, as there does not seem to be a tendency of either a fast or slow connection.

6.2.5 Tunnel spawn limit

The current implementation spawns two threads for each tunnel. Does this cause any practical limit? Will it be limited by the amount of threads, that a process can spawn, or the amount of ports to listen on?

For the test, I set up the MacBook to send `TunnelRequest` in a loop to the Mac mini. The Mac mini stops at exactly 750 tunnels and then fails. Looking at the error code of 24, I get the message "Too many open files". Which means that the threads are likely not the limiting factor, but rather the amount of open files the operating system allows for a process. 750 is likely more tunnels, than would practically be used from a single super-node. The speed would also suffer, if it was still a 10MB/s connection, like previous test. From a security point of view, it's rather easy to fill the 750 in a matter of seconds, but it does not allow for anything else, but denial-of-service. If it becomes an issue, the network operator would be able to see the domain of the node and manually exclude it from the network. Alternatively, the code could set a limit for each node only opening one tunnel and each domain is allocated a fixed amount of tunnel per super-node.

6.3 Reflection

Although the tests are carried out in a network environment, which is far from the real use-case, we still get a look at how it performs.

We got to see the amount of messages that a node can handle when loading the CPU to the limit. I'm not sure what I expected, but it does not seem to be a particularly low limit. Requiring to process just around $100,000$ messages every second, seems almost unachievable in a normal use-case.

Encryption would possibly add more load on the CPU and limit the amount of messages being sent. It is hard to estimate an exact amount, but a well-thoughout encryption scheme would likely still deliver messages in several ten thousands per second. Ultimately, non-secret commands could disable encryption, if it gives a high performance impact – for example `GetSuperNodes`.

There does not seem to be any surprises in any of the tunnel tests. The throughput of tunnels are about the same as a direct connection and the amount of tunnels that can be spawned is not a restricting factor. Although the limiting factor would still be the bandwidth; which is why we need the super-nodes to distribute onto.

None of the tests resulted in the node-software crashing at any point. This shows, that the policy of discarding messages, which causes an error, at least keeps the system running and makes a quick recovery possible. It also does not stop what a node is already doing, as it could if a session was disrupted because the node's bandwidth is filled.

6.4 Partial Conclusion

Performance and stability are essential parts of the project.

The way I've decided to handle errors, by just leaving an action if it fails, seems to be a viable solution. This adds to the system's tolerance to errors on both the sending and receiving end.

On the practical test of the software, I've

shown, that with minimal hardware, it's still possible to get a quite high throughput. Although encryption, which is not implemented, might give a performance penalty, it still seems to be yielding good results. With a throughput of more than $100,000$ messages/sec, it seems to have some head-room for future performance-heavy features.

Throughput on tunneling and the amount of tunnels a node can handle seems to be as expected. The throughput is limited by the bandwidth of the connection and with support for opening several hundred tunnels at once, it seems to be promising.

7 Conclusion

The Internet does not live in the same way, as it was envisioned back in its early days. When it was first created, everything was connected and problems, like hacking and exposing private information, was not a concern. Today, the Internet is a hostile place, where everything has to be locked away. This development has restricted certain parts of the Internet. With the future of IPv6, it's not certain, if some of these values will be resurfacing, or if the world has become dependent on the restrictions we have.

Major parts of the project circled around scalability, while still acheiving the goal of interconnecting end-devices. A quick solution could easily be made, to proxy everything we needed, but it did not scale well. The use of peer-to-peer (P2P) networking, allowed us to not have a linear relationship between load and the need for resources at a central server.

A P2P network has proven to be a viable solution for scalability for file-sharing. Some of these experiences and technologies might have a future in other fields of study. A P2P network leverages on the effort of many, to serve a single purpose. It might in the big picture use more resources, but it allows for everyone to participate.

Traditionally, P2P networks has focused on decentralization and resiliance to censorship. These are of course possible goals of P2P networks, but projects like this and Skype, shows that more 'legitimate' businesses also can gain from the technology.

Trust is a part of the Internet, which has disappeared. With the hostility of the Internet, it has become a central subject of any Internet-based project, to consider the risks of hackers. For this project, I've taken precautions throughout the design-phase, to make the system inherently cautious against a range of common attack vectors.

The use of cryptography has allowed for the system to bridge the gap between sharing data and trusting the sender. By signing every piece of communication, any node will be able to validate its authenticity completely independent from others. It's one thing to have a Distributed Hash Table (DHT) between a large amount of nodes, and having to design a DHT which is resilient to tampering and keeps its integrity.

The stateless nature of the design, allows for great scalability and fault-tolerance. It's an inherent problem of P2P networks, that any node can go offline at any time, unlike traditional server, which we assume is online when we need it. Instead of building around this instability, we make it part of the system, which gives us the gain of easily recovering from errors in almost any circumstance.

The experince of learning Rust and implementing the system from scratch, seems to have been worth the effort, when look at the performance. Although the performance have not been compared to alternative implementations, it seems like positive results, when a single laptop has the ability to handle around $100,000$ messages/sec before dropping packets. With the vision of the system, it will possibly allow for scalability in the millions of users, if the super-node structure is holding up.

8 Reflection

What would be the future of this project? Sadly, looking from a completely objective point of view, this project is a band-aid on a fractured bone. This project tries to accomplish something, which should be solved by the underlying design of the Internet. The effort, which I put into circumventing security features, might have been better used on fixing the underlying problem: The Internet itself. The Internet grew from a very technical perspective of "what is possible". Now that we have prooved it works, maybe we should start to look at what we *want* from the Internet.

Does it really have to be as hard as rocket science to allow for yourself to access your webcam from work? Could we possibly engage the user in IT-security in a light fashion?

We have the Internet, which is inherently hostile, but we do not have a one-size-fits-all cure for solving the problem. It's completely up to the individual server host to make sure to enable encryption, because it's a technology which has been patched on-top. Maybe we need technologies like PGP and a web-of-trust to be more influential on a low-level IP-basis. It would vastly change the view of IT-security, if every household and company, could manage access to its network by 'simply' signing an end-user's certificate. It does not have to be a centralized government-control system, but rather in the line of the way key-based authentication works on SSH. You construct a key and tell people about it.

Until the underlying problem is solved, there might be a place in the world for a system like this. Given the right resources, we might see a project similar to this in the futuristic Internet-of-things. When you want to turn on the air-conditioner while driving home, or turn on the toaster from work, you'll need some way to connect to these devices. The way to solve it today, will be up to the individual manufacturer of the device. One would hope this does not lead to fragmentation, where you need one website or program to control each device. The need for standardized ways to interconnect these things, will likely grow by the years.

9 References

[1] Bittorrent protocol specification v1.0. https://wiki.theory.org/BitTorrentSpecification#Identification. [Online; accessed on 13th of June 2016].

[2] Can every grain of sand be addressed in ipv6? http://skeptics.stackexchange.com/questions/4508/can-every-grain-of-sand-be-addressed-in-ipv6. [Online; accessed on 13th of June 2016].

[3] Cargo guide. http://doc.crates.io/guide.html. [Online; accessed on 13th of June 2016].

[4] Eric Bangeman. Study: Bittorrent sees big growth, limewire still 1 p2p app. http://arstechnica.com/uncategorized/2008/04/study-bittorren-sees-big-growth-limewire-still-1-p2p-app/, 2008. [Online; accessed on 13th of June 2016].

[5] Salman Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *CoRR*, abs/cs/0412017, 2004.

[6] Ernesto Van der Sar. Pirate bay loses new domain name, hydra lives on. https://torrentfreak.com/pirate-bay-loses-new-domain-name-hydra-lives-on-150522/, 2015. [Online; accessed on 13th of June 2016].

[7] The Rust Project Developers. The rust programming language. https://doc.rust-lang.org/book/README.html. [Online; accessed on 13th of June 2016].

[8] The Rust Project Developers. The rust reference. http://doc.rust-lang.org/reference.htm. [Online; accessed on 13th of June 2016].

[9] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[10] John Gallagher. Rusqlite. https://github.com/jgallagher/rusqlite. [Online; accessed on 13th of June 2016].

[11] Network Working Group. Internet protocol, version 6 (ipv6) specification. https://www.ietf.org/rfc/rfc2460.txt, 1998. [Online; accessed on 13th of June 2016].

[12] Craig Heffner. Exploiting network surveillance cameras like a hollywood hacker. https://www.youtube.com/watch?v=B8DjTcANBx0. [Online; accessed on 13th of June 2016].

[13] Network World Inc. Types of Firewalls. http://www.networkworld.com/article/2255950/lan-wan/chapter-1--types-of-firewalls.html. [Online; accessed on 13th of June 2016].

[14] Petar Maymounkov and David Mazières. *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*, pages 53–65. Springer Berlin Heidelberg, 2002.

[15] Microsoft. Test IP-to-MAC Address Resolution with ARP. https://technet.microsoft.com/en-us/library/cc961394.aspx. [Online; accessed on 13th of June 2016].

[16] Microsoft. What is nat? https://technet.microsoft.com/en-us/library/cc739385%28v=ws.10%29.aspx?f=255&MSPPError=-2147217396, 2015. [Online; accessed on 13th of June 2016].

[17] Marie Ravn Nielsen. Sn fungerer den ulovlige filmtjeneste popcorn time. http://www.dr.dk/nyheder/kultur/film/grafik-saadan-fungerer-den-ulovlige-filmtjeneste-popcorn-time. [Online; accessed on 13th of June 2016].

[18] OWASP. Fuzzing. https://www.owasp.org/index.php/Fuzzing. [Online; accessed on 13th of June 2016].

[19] Edwin Schouten. 5 cloud business benefits. http://www.wired.com/insights/2012/10/5-cloud-business-benefits/. [Online; accessed on 13th of June 2016].

[20] The Rust team. Rust 1.0: Scheduling the trains. http://blog.rust-lang.org/2014/12/12/1.0-Timeline.html. [Online; accessed on 13th of June 2016].

[21] Iljitsch van Beijnum. With the americas running out of ipv4, its official: The internet is full. http://arstechnica.com/information-technology/2014/06/with-the-americas-running-out-of-ipv4-its-official-the-internet-is-full/, 2014. [Online; accessed on 13th of June 2016].

[22] Liang Wang and Jussi Kangasharju. Measuring large-scale distributed systems: case of bittorrent mainline dht. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013.

Appendices

A Hitting the malicious super-node

This table shows the chances of querying a malicious super-node. The table shows only the chances for queries without redundancy. If one were to use a redundancy of 2 and the total amount of malicious super-nodes were less than 2, we could be the pigeonhole principle determine, that a query would still succeed.

Total Super-nodes/ Malicious Super-nodes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0,00	1,00															
2	0,00	0,50	1,00														
3	0,00	0,33	0,67	1,00													
4	0,00	0,25	0,50	0,75	1,00												
5	0,00	0,20	0,40	0,60	0,80	1,00											
6	0,00	0,17	0,33	0,50	0,67	0,83	1,00										
7	0,00	0,14	0,29	0,43	0,57	0,71	0,86	1,00									
8	0,00	0,13	0,25	0,38	0,50	0,63	0,75	0,88	1,00								
9	0,00	0,11	0,22	0,33	0,44	0,56	0,67	0,78	0,89	1,00							
10	0,00	0,10	0,20	0,30	0,40	0,50	0,60	0,70	0,80	0,90	1,00						
11	0,00	0,09	0,18	0,27	0,36	0,45	0,55	0,64	0,73	0,82	0,91	1,00					
12	0,00	0,08	0,17	0,25	0,33	0,42	0,50	0,58	0,67	0,75	0,83	0,92	1,00				
13	0,00	0,08	0,15	0,23	0,31	0,38	0,46	0,54	0,62	0,69	0,77	0,85	0,92	1,00			
14	0,00	0,07	0,14	0,21	0,29	0,36	0,43	0,50	0,57	0,64	0,71	0,79	0,86	0,93	1,00		
15	0,00	0,07	0,13	0,20	0,27	0,33	0,40	0,47	0,53	0,60	0,67	0,73	0,80	0,87	0,93	1,00	
16	0,00	0,06	0,13	0,19	0,25	0,31	0,38	0,44	0,50	0,56	0,63	0,69	0,75	0,81	0,88	0,94	1,00

Figure 25: The changes of hitting a malicious super-node without redundancy

B Command Types

The messages can carry a range of commands. The command-types are shortly described below.

Name	Requester	Handler	Meaning
GetOperatorNodes	Any	Any	Request list of operator-nodes
OperatorNodes	N/A	Any	List of operator-nodes
GetSuperNodes	Any	Any	Request list of super-nodes
SuperNodes	N/A	Any	List of super-nodes
SubmitSuperNode	Any	Operator	Used by a node to become super-node
TransferPartition	Super/Operator	Super/Operator	Used to transfer all information for a key-space parition on the DHT
GetNode	Any	Any	Sent before starting a tunnel or returned when contacted by an unknown node
Node	N/A	Any	A node structure, which the receiver saves
GetNodeInfo	Any	Any	Request information about a node
NodeInfo	N/A	Any	Informatio about a node
GetTunnelNodes	Any	Any	Request a node to return all the nodes, which has the specified location.
TunnelRequest	Any	Tunnel	Requesting a TCP-tunnel to be opened
TunnelOpened	N/A	Any	Returned when TCP-tunnel gets open
Ping	Any	Any	Only available in test-build. Checks if node is online
Reping	N/A	Any	Only available in test-build. Response to Ping

C Threat assestment

I will go through the risk assessment of the five most likely attacks. The attacks are ordered in increasing severity. If we as an attacker accomplishes one of the first four attacks, the previous ones will be trivial to execute afterwards.

Threat	Likelihood	Severity	Consequence	Mitigation
Stealing Data	Medium	Low	Harassment of individuals. Monitoring users in broad.	Move away from public super-nodes
Controlling the Network	Medium	Medium	Outage of the system. No permanent damage to users or the system. Damage to business	Ban domains, move away from public super-nodes
Stealing the Operator's Key	Low	High	Complete control of the network	Revoke private key and reinitialize network
Code Injection	Low	Very High	Escalating the attack out of the bounds of the network. Spread the malware through the P2P network.	If done right, it's game over. Patch and update software
Unauthorized Tunneling	Medium	Very High	Possible escalation of attacks. Access to unprotected servers.	Patch and update software

C.1 Stealing Data

Gaining access to secret data can be valuably, in case of a website or a business, because they will store usernames, passwords, emails, credit cards and so on. This system does not have any passwords or valuable information at all. In fact, all the information we have is already shared across the super-nodes. The "passwords" I use are cryptographic keys, which are assumed to be nearly impossible to replicate even when the public key is shared on the P2P network.

In a longterm plan, an attacker could take the role of a super-node and collect data as part of a plan. Unless the hacker has a specific target in mind, I cannot see a viable end-goal. Read more about privacy in section 4.1.

C.2 Controlling the Network

A possible goal would be to take control of the network. The benefit of this would mostly be to collect data, as suggested above. An alternative goal would be to disrupt part of the network or the whole network.

For specific ways to disrupt the network, see section 4.3.

C.3 Stealing the Operator's Key

Besides the next case of code injection directly into the P2P software, it's not expected that the central server is hacked directly. A server like this should be isolated and only run the task of being the operator-node. All other services like websites and mailserver should be contained elsewhere to minimize the risk.

A possible attack would be to spearphish a sysadmin or developer, which has remote access to the server. This would likely be the weakest link and enable access into the server to fetch

the private key.

As described in section 4.3, if one were to gain access to an operator-node's private key, the attacker would be in complete control of the network. But would be limited to the functionality which the network allows.

C.4 Code Injection

The most desirable of all the goals for a hacker, in the remote code injection. If one were to find a way to execute arbitrary code just by sending a message to a node, it be highly valueable.

Arbitrary code execution enabling an attacker to gaining absolute control of anything on the network. It would be possible to fetch a list of any accessible node and install almost any type of malware.

C.5 Unauthorized Tunneling

Apart from code injection, the second worst case would be unauthorized access to end-devices. I can see it happen at three different levels.

First, foreign user accesses end-device. This would possibly happen by circumventing the check of which domain both nodes are from. Or through making the operator sign a public key for the wrong domain.

Second, local user accesses end-device which isn't registered. This could possibly happen, if an attacker can change the IP-address of an end-device's datastructure without breaking the signature. Alternatively, it could be by making a legitimate connection to an end-device, but using an exploit on the end-device to start new connections.

Third, foreign user accesses any device on local network. This is a combination of the previous two. Depending on the way the exploit works, having the previous two might not open up for this one.