Bootstrapping P2P VPN

Felix Weißl
Betreuer: Benjamin Hof, Lukas Schwaighofer
Seminar Future Internet SS2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: felix.weissl@tum.de

ABSTRACT

Decentralized or P2P (peer-to-peer) VPNs (virtual private networks) are popular due to their advantage over centralized VPNs or the classic client-server model. This paper surveys the main difficulties of P2P VPNs in general, but most importantly of bootstrapping a new peer in such a network which is the mechanism of a new peer finding and joining it. Different solutions are indicated on existing implementations and evaluated separately. Especially the two contrasting models of supernodes versus the joining over an existing public overlay are examined for they are most commonly deployed in existing P2P VPNs. It becomes apparent that the presented solutions only partly cover the needed functionality, especially in terms of security and the traversal of NAT devices.

Keywords

peer-to-peer, virtual private network, boostrapping, decentralized overlay, NAT traversal

1. INTRODUCTION

Peer-to-peer networking has gained a great interest in the last years due to many popular applications in file sharing (BitTorrent, Gnutella, KaZaA), but also media streaming or voice-over-IP services such as Skype. Its main advantage over the client-server model is that no central authority such as a server is needed for sharing resources because every node (peer) is taking part in the network actively, by meaning of relaying data for other nodes.

A virtual private network (VPN) provides an illusion of a local area network by creating secure and authenticated communication links amongst its participating nodes over the global Internet [23]. Today VPNs are frequently used, e.g. in companies so that employees can enter the intranet securely without beeing physically linked to it. But they are also attractive to home users who can establish a secure end-to-end connection with each other for sharing data or other local resources.

A P2P VPN - as the name suggests - combines these concepts and therefore offers a completely decentralized overlay network. Operations that work for client-server model have to be rethought and adapted to a distributed overlay model.

The rest of this paper is organized as follows. Section 2 gives short background information and definitions about the main terms used later in this work. Section 3 explains the key requirements of a P2P VPN and two main problems of the bootstrapping mechanism: peer discovery and NAT traversal. Existing implementations and how they solve these issues are discussed in Section 4. Other interesting or notable approaches are shown in Section 5 while Section 6 presents concluding remarks.

2. BACKGROUND

A P2P system relies on a network topology where the nodes are connected to each other and have the purpose of sharing resources such as storage, bandwith or CPU cycles. They form a self-organizing overlay without the need of a centralized server or authority [18, p. 6].

An *overlay* network is defined as an application layer virtual or logical network in which end points or nodes are addressable and that provides connectivity, routing and messaging between these nodes [18, p. 7]. An overlay is layered on top of an existing layer, called underlay, which in our context is the global Internet [3].

In *centralized* VPNs all the communication flows through one server which also provides bootstrapping and authentication. Two peers never directly send packets to each other because there is always a central authority such as a dedicated server relaying the traffic between them. Of course, this server can become a single point of failure which is a problem for high availability networks. A popular implementation of this technique is OpenVPN [15].

In *decentralized* networks however, there is no distinction between clients and servers since every peer in the network is equal. Routing, bootstrapping and authentication is done directly with each other and as a result must be explicitly defined [23]. By decentralizing the network, bandwidth and computation can be distributed between the peers.

In *unstructured* P2P systems a node relies only on its adjacent nodes by the terms of routing and forwarding. The graph is randomly created depending on which nodes are joining or leaving and their interactions. That makes joining and leaving rather easy, but has a problem in searching, e.g. a file, because the request has to go through the entire network in the worst case. Two popular implementations are n2n [13] and P2PVPN [6] which are discussed in Section 4.1.

Structured P2P systems provide distributed lookup services by forming a self-organizing topology such as a ring [23],

doi: 10.2313/NET-2014-08-1_01

a tree or a mesh structure where every node has a link to all other nodes in the network. These lookup services are provided by so called *distributed hash tables (DHT)*. They are decentralized data structures, in which keys with associated data are mapped to specific node IDs in the overlay [21]. DHTs are typical for structured P2P systems and commonly used in P2P architectures.

3. REQUIREMENTS AND DIFFICULTIES

Before introducing existing P2P VPN implementations this Section first mentions certain requirements that have to be fulfilled and difficulties that have to be dealt with.

3.1 Requirements

The main design goals mentioned in [8, 9, 2] for a purely decentralized bootstrapping mechanism to fullfil are:

1. Availability: Every peer must be able to join or leave the network when it desires to. That means the bootstrapping mechanism must be guaranteed at any time which makes an approach with an unstable success rate insufficient [8]. A single point of failure should always be avoided as far as it is possible for decentralized networks. That is also ensured by equally distributing workload between the peers so that an attacker cannot compromise the availybility by initiating DDoS attacks on them.

2. Automation and Self-organization: For a desired real world applicability a peer has to be able to join the network with as little as possible manual interaction. Furthermore the bootstrapping mechanism should not have to rely on prior knowledge, e.g. a list of bootstrap servers that a joining node needs to have stored [2].

3. Efficiency: Although our main focus in this paper is the connection establishment, it is important to keep the time period for that to a minimum.

4. Scalability and Robustness: The bootstrapping mechanism has to work in small private as well as in large communication networks involving millions of peers. If certain peers malfunction or leave the network rapidly, there have to be procedures changing the network back to its full functionality. In this context "churn" is the rate of joining and leaving peers of a network [1].

5. Security: Trusting each other in P2P systems is harder than in centralized systems where authorities can interact as a trusting party such as a PKI. Therefore is important to prevent bad peers from joining the network.

3.2 Finding a peer

The initial problem for a peer before connecting to the P2P VPN is to discover the network and an active participant as an entry point. We assume that a previous connection has never been established so it is not possible to own some kind of peer cache.

As a first approach one could integrate a list of known peers into the P2P software itself. Gnutella for instance, a very large decentralized peer-to-peer network, uses this approach [5]. Of course these peers must have a high availability and become target for attacks. Internet providers could use this information to prevent their clients from joining such a network. The peer list, in this context often called hostlist or webcache (Gnutella) [7], could also be shared and requested on a public HTTP server. But this would end up in a partially centralized architecture, so this is not an optimal solution. The second approach demonstrated in [5] uses brute-force scanning over a list of promising IP addresses. The list is a result of a special heuristic using statistical profiles using the IP ranges of start-of-authorities in the domain name system [5]. This method requires that developers can get a list of IP addresses for the network and that the peers use the default port for the protocol. A similar approach is presented in Section 5.

3.3 Middlebox Traversal

Another technical issue with P2P VPNs are middleboxes such as *Network Address Translators* (NATs) and *firewalls*. A firewall is used to filter incoming and outgoing traffic and is set up on the edge of the global Internet to a local network for protection. The firewall's filter rules are usually designed to block incoming connections that are not a response to a previous outgoing request. That is a problem for P2P networks where a new peer wants to communicate with another peer behind such a device for bootstrapping. NAT devices on the other hand help to overcome the shortage of global IPv4 addresses by hiding a local network behind just one globally routable IP address. All clients in this local network are given private IP addresses (e.g. in 192.168.0.0/16) and every packet's local IP and port are translated to a global IP and port. That means that local clients cannot be addressed directly with a unique IP before the peer behind the NAT initiates the connection and the NAT can route the packet properly. The ongoing migration from IPv4 to IPv6 does not necessarily resolve this issue. Although NAT is not needed anymore and peers could directly communicate, firewalls would still be essential for blocking unwanted, incoming connections. Also since IPv4 is still deployed, NAT will be as well.

It can be assumed that the majority of peers in P2P networks are home users and behind a NAT device or a firewall as well. Thus *NAT traversal* mechanisms are inevitable for operating a P2P VPN.
A simple mechanism for a host behind a NAT ("NATed" host) would be to configure the NAT device yourself and enable port forwarding, so that all connections can traverse on a negotiated port. Because the majority of users would not have the rights or knowledge to do this, this is not really an option. The common NAT traversal mechanisms used in current implementations are elaborated on the subsequent Sections.

3.3.1 TURN: Traversal using Relay around NAT

TURN [10] is very reliable and deployed in centralized VPNs. If two NATed hosts want to communicate, each of them has to initiate a connection with a well-known globally routable server. This TURN server relays every message between both hosts and therefore also passing both NAT devices. Of course this is a very inefficient and resource-intensive method and only scalable for small networks. But since this

doi: 10.2313/NET-2014-08-1_01

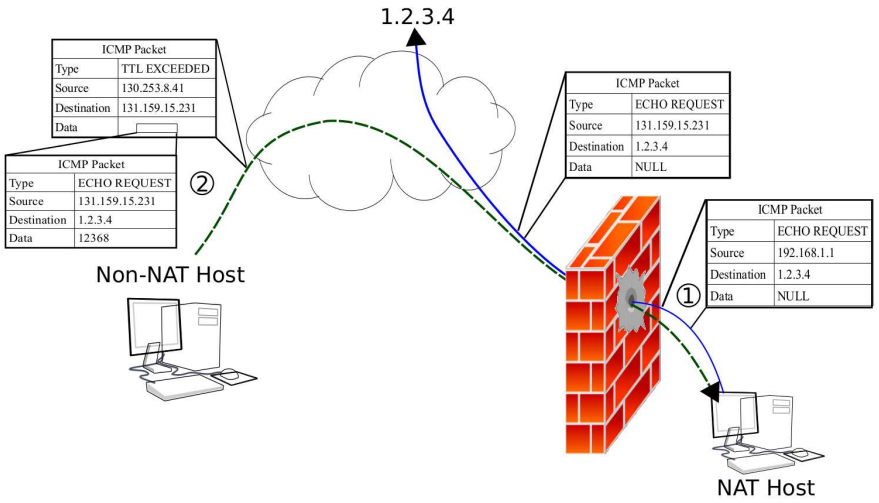

Figure 1: Here *autonomous traversal* of a NAT device (represented as wall) is shown. The solid line describes outgoing and the dashed line incoming ICMP packets. Figure source: [11]

is the only method working on all existing NATs relaying is a useful alternative if other traversal methods fail [4].

3.3.2 UDP Hole Punching

"Hole punching" is one of the simplest and most practical NAT traversal techniques [4] for the scenario that two hosts are behind a NAT device. Both NATed hosts have to have an active UDP session with a so-called rendezvous server. First both hosts exchange their private and public endpoint information (local and global IP:Port) via the rendezvous server as an intermediary using the Session Traversal Utilities for NAT protocol (STUN) [16]. By this point the server is not needed anymore. Both hosts try to establish a UDP connection between each other on both endpoints. Considering the simple scenario of both hosts residing in the same network, the connection can be established. In the scenario with two different NATs, the first outgoing message of both hosts "punches a hole" through their own NATs. That means if the NAT is well-behaved, it will preserve the hosts private endpoint and allow incoming traffic and address translation [4]. Well-behaved NATs in this context are *cone* NATs, which support consistent endpoint translation, and not *symmetric* NATs, where "hole punching" is not successful [4]. Tests on a wide variety of deployed NATs have shown that about 82% support hole punching using UDP [4].

3.3.3 TCP Hole Punching

Since there are VPNs relying on TCP such as P2PVPN (Section 4.2.1) hole punching is needed for TCP as well. Compared to UDP the idea is very similar, however since TCP is a stateful protocol, there are some issues. For a successful hole punching both NATs need to be able to establish a connection with the rendezvous server and the other hosts NAT device over the same port. Both connections need a TCP socket which binds to the specific port and this fails if the port is already bound to another TCP socket [4]. Additionally another socket is required to listen for incoming connections at the same time. Since this is not possible with todays Berkeley (BSD) sockets, the option SO_REUSEADDR must be used which allows multiple socket bindings to the same local endpoint [4].

The NAT traversal procedure is as follows. Both NATed hosts use their active TCP session with the rendezvous server to exchange their public and private endpoints (local and global IP:Port) similar to Section 3.3.2. Using the same local TCP ports that both hosts used with the rendezvous server, they keep sending messages to each other's private and public endpoints while listening for incoming messages [4]. Once the outgoing messages succeed and incoming messages can use the created "hole", the connection must be authenticated to make sure that the other host is really the desired one. In the same test environment as before with UDP, TCP hole punching is supported by about 62% of deployed NATs [4].

3.3.4 Autonomous NAT traversal

A profoundly different method proposed by Müller et al. [11] traverses NAT devices without any third party. We assume that there is a host A behind a NAT device and we are aware of its global IP address, e.g. obtained from a peer list. If an external host B (not behind any NAT device) wants to communicate with this host A, only two messages are required:

1. The NATed host A continuously sends ICMP ECHO REQUESTS to an unallocated IP address, such as 1.2.3.4 [11]. The expected incoming ICMP DESINATION UNREACHABLE packets can be ignored.
2. Because the NAT device now routes corresponding reply messages, the external non-NAT host B sends a fake ICMP TTL_EXPIRED message, which the NAT host A listens for. Such a messages doesn't have to have 1.2.3.4 as source address to be transmitted [11]. Therefore B is the source and the port for future messages is added through the payload as a new ICMP

packet. As A is now aware of B's IP global IP address, it can initiate the actual connection.

A more descriptive representation can be seen in Figure 1 with the two messages numbered likewise.

An implementation of this relatively new approach is the *pwnat* tool by Samy Kamkar. It is also implemented by GNUnet as a transport plugin [12]. Unfortunately for sending these ICMP messages, peers need superuser privileges which may not be available. Müller et al. also propose an alternative method using UDP packets instead of ICMP ECHO REQUESTs which works better on some NAT implementations. Through testing this combined traversal method on a large number of NAT implementations it has shown that the success rate quite remarkable averaging at 82% [11]. Unfortunately this approach has a very little success rate if both hosts are behind NAT devices.

Summarizing this Section, none of these traversal techniques work for every NAT implementation and most of them require a third party. Therefore systems typically implement more than one technique for high success rates [11].

4. CURRENT APPROACHES

There are several approaches which meet the requirements of the previous Section 3 to some extent, but not in its entirety. This shows that this topic still needs research in the future. In this context we distinguish two common bootstrapping methods - using supernodes and existing public overlays. Other notable approaches which do not fit in this Section are discussed in Section 5.

4.1 Supernodes

This model is not a purely decentralized VPN model in which no participant has significantly more features or workload than the others. As implied in Figure 2 these *supernodes* or *superpeers* build a backbone overlay [1, p. 64] and are deployed to deal with the issues of bootstrapping, NAT traversal and routing between the other edge nodes.

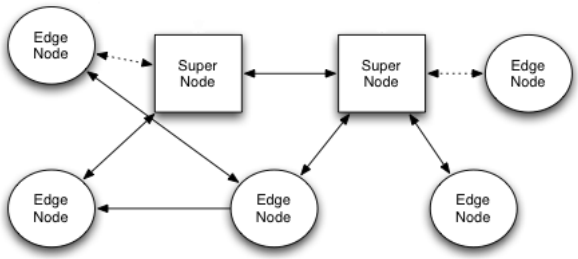

Figure 2: This shows an overlay architecture where supernodes are used for a variety of services, including bootstrapping. Figure source: [13, p. 6]

4.1.1 N2N

N2N is an unstructured layer-two over layer-three P2P VPN [13] released open-source. The software comes with two binaries, the client for edge nodes and the server for supernodes. Technically both can also run on the same physical machine. To create the network, there have to be one or more supernodes which must be directly addressable, that is without a NAT device in between. Usually nodes with high bandwidth and computing power are used as supernodes. Each supernode has to set up communication with one other supernode. The same applies to new edge nodes which require to know the global endpoints of a supernode to join the network. For peer discovery supernodes use layer-two broadcast and they also forward broadcast and multicast packets of the other supernodes to their edge nodes. The result is that each edge node receives these messages and can build up a peer list ({MAC, UDP socket} pair) of the network. As UDP socket we describe the respective endpoint information, that is IP and port.

As previously mentioned there could be middleboxes (see Section 3.3) between supernodes and edge nodes. For the NAT traversal N2N uses UDP hole punching (see Section 3.3.2). Each edge node uses this list to start the so-called peer registration [13] in which it sends out registration requests directly to the other edge nodes. If this UDP hole punching is not successful the specific NAT implementation is probably not supported. If both edge nodes have *symmetric* NATs, the usual NAT traversal is not possible. That is because the global endpoints which are used between an edge node and a supernode cannot be reused with another edge node instead of the supernode. Symmetric NATs would use a different external mapping [13]. In this scenario the supernode works as a relay between the two edge nodes (see Section 3.3.1).

However, N2N is quite inconvenient for end users because bootstrapping and supernode set-up is not automated and has to be deployed by the edge nodes. N2N *1.x* also has some security limitations mentioned on its project site such as the lack of authentication or its vulnerability to replay attacks due to missing nonces in its encryption. A second version with enhanced security extensions is still in developement [13].

Regarding the requirements in Section 3.1 it is questionable if N2N VPNs are scalable with a high amount of users due to the ongoing broadcast and multicast messages.

4.1.2 Tinc

Tinc VPN [19] is more decentralized than N2N because it is organized as a mesh network in which every node relays data for other nodes where it is needed [19]. This may sound similar to supernodes in N2N but every peer has the same functionalities here. The drawback is that this network lacks self-organization and requires explicit specification of which links a node has to create [23]. That means the network has to be organized by hand to be able to deploy tinc's routing protocols [22]. Furthermore tinc doesn't work well if nodes leave. If the missing node was the only one keeping the network together, the network splits in two different ones [22].

4.2 Public Overlays

Another method proposed in [21] is to bootstrap an own private overlay by using existing public overlays with a high availability. This is especially interesting for home users for

doi: 10.2313/NET-2014-08-1_01

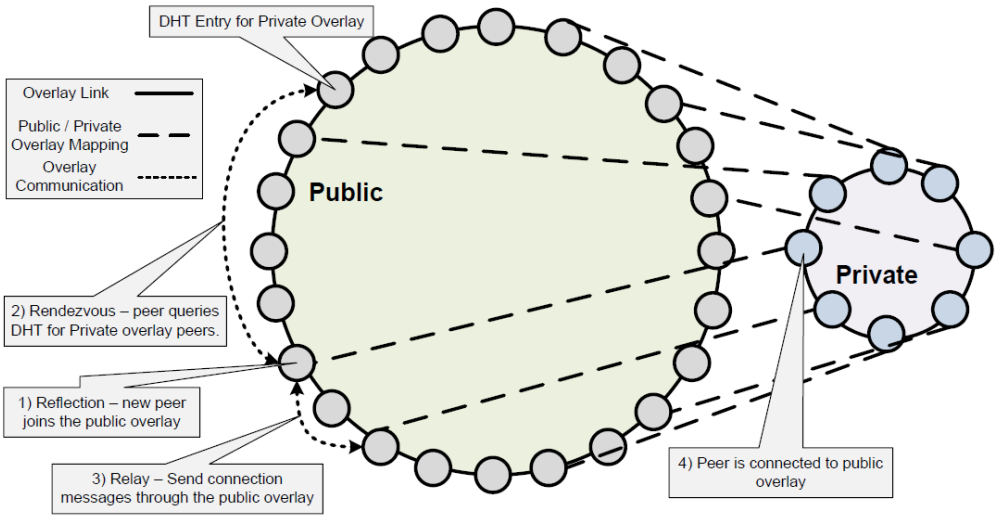

Figure 3: A public overlay is used to bootstrap a private overlay. Figure source: [22, p. 3]

which the configuration and set-up of own rendevouz servers or supernodes may be too difficult or costly.

4.2.1 *P2PVPN*

The application "P2PVPN" is a decentralized unstructured and open-source VPN [6] and is primarily designed for home users, offering a GUI alternative to N2N and claiming equal funcionality. For bootstrapping it uses the filesharing protocol BitTorrent as a public overlay and entry point. The BitTorrent network is a very popular and large-scale P2P network. The first peer creates the overlay by registering a virtual file on a so-called tracker which works as a rendezvous point now [21]. Following peers who must be aware of its hash value and can obtain the peer list from the tracker. If a tracker goes offline, a different one has to be used. But P2PVPN also offers users to disable the BitTorrent option and bootstrap a purely decentralized network themselves. In this case at least one peer has to be globally routable and must serve as entry point. The first peer can now generate so-called invitations, which are similar to certificates used in the TLS protocol, and only valid for a single peer. They contain all information needed for an external peer to join the network either with the use of a BitTorrent tracker or the direct way with IP and port. The user only has to specify the lifetime of this peer, the other values (signatures, keys) are generated by P2PVPN. These invitations have to be transmitted over a secure channel manually. For peer authentication P2PVPN uses RSA key pairs; inside the network traffic stays encrypted with AES using CBC mode.

Unfortunately NAT traversal is not provided at all and own port forwardings must be configured by hand. Furthermore it has been evaluated that P2PVPN's routing algorithms do not scale well and the software may have problems with large networks [6, p. 61]. That is because update messages between the peers contain their whole routing table.

4.2.2 *GroupVPN*

In [21] two possible public overlays, XMPP and Brunet, are discussed and extended to the decentralized structured and P2P VPN model called GroupVPN. The *Extensible Messaging and Presence Protocol* (XMPP), also known as Jabber, is a popular XML based messaging protocol [17] which is used in many popular overlays such as GoogleTalk (until 2013) or Facebook Chat [21]. Brunet on the other hand is a freely available DHT (see Section 2) similar to the defunct OpenDHT which was running on the global research network PlanetLab using the Symphony protocol, therefore creating a dedicated bootstrap overlay [21]. A bootstrap overlay consists of one or more bootstrap resources which at least one is publicly available for users with the desire to join. After requesting so called bootstrap nodes from one of these resources it assists in connecting the user with other nodes in the new joined overlay.

The bootstrapping scheme is similar to "P2PVPN" (see Section 4.2.1). For a better understanding Figure 3 is quite helpful: First a node connects with the public overlay and exchanges public endpoint information. This is done using the STUN protocol (compare Section 3.3.2). The next step is to look up the entry of the desired private network in the public networks DHT. Now another peer, which is in both networks, is used for joining the private network.

5. OTHER WORK

This Section mentions approaches of bootstrapping that were not being implemented in P2P software. Despite this, they deal with the bootstrapping problem in a different way as it could be seen in the previous Sections.

In [2] the authors describe an approach of *Local Random Access Probing* where a peer first joins a "bootstrap P2P network" to bootstrap into the actual desired network by name lookup. Of course the problem is just shifted to bootstrapping into this "bootstrap P2P network". That is done by a variant of the existing *Random Access Probing* where a

peer sends messages to random IP addresses until it finds a node which is already part of the bootstrap P2P network. As a difference to that only address ranges around the current IP address of the user are probed. The idea is that private users with Internet access via dialup networks (DSL, cable) would more likely use P2P applications and can profit from an increased communication speed [2]. As an example the eDonkey filesharing network, usually port 4662, is tested in the German IP address space, considering only dial-up networks since the probability for them running such a protocol is higher than others. As a result bootstrapping is successful after around 600 probes which averages in an overall time of 20 seconds. This approach is promising and fulfills most of the requirements of Section 3. The drawback is of course that such a large "bootstrap P2P network" such as eDonkey has to exist in the first place. Furthermore TCP SYN port scan is used as a probing mechanism which requires that the network has a unique port that every peer uses for the protocol.

Another approach uses the *Internet Relay Chat* (IRC) [14], an open, decentralized network of chat servers with high availability [9]. IRC is interesting because once connected to one server of an IRC network, one could communicate with any other client in the same network. Knoll et al. [9] provide a decentralized bootstrapping mechanism for IRC:

1. Connect: Choose an arbitrary IRC server of a specified network and join the chat room, a so-called channel, as a new user with a chosen unique nick name.
2. Discover: If bootstrapping peers (e.g. nick name format "bs_peer") are available in the channel, proceed with the request. If not, this peer becomes a new bootstrapping peer, meaning that a new overlay is created here.
3. Request: The peer requests a peer list and one of the bootstrapping peers responds with a "bootstrapper" list. This list contains contact information of the actual bootstrapping peers. This way the ones that are in the IRC channel are not necessarily the same as the ones that are directly requested outside the channel.
4. Overlay join: With the given list the channel can be left and the peers can be contacted directly. Because these could be hidden behind a firewall or NAT device, a NAT traversal mechanism based on the STUN protocol using supernodes is used [20].

Their approach does not focus on the security and privacy aspects but on the practicality, e.g. an attacker could masquerade as one of the bootstrapping peers and send own lists to new peers.

A second approach by Knoll et al. [8] uses *DDNS*, a dynamic variant of the *Domain Name System*. Dynamic DNS providers offer clients to to map their IP addresses to a (sub)domain name via a password-protected web interface. The idea is that one bootstrapping peer is always mapped to this domain. Clients outside the network can request a peer list using the domain name which is the only thing they have to know about the network. If the current bootstrapping peer leaves, one of the other peers replaces it with its own IP address. The evaluation shows that this approach works surprisingly well with an increasing peer number and with low workload on the DDNS provider (IP address changes). The drawback of this approach is that there always has to be one peer providing the only entry point for the network and thus getting more traffic than the other peers. Furthermore the password for the web interface has to be some kind of well-known which would become a problem if a malicious peer changes this password in the web interface.

6. CONCLUSION

This seminar paper observed the bootstrapping problem and possible solutions. When evaluating existing implementations, not one could fulfill every requirement specified in Section 3.1, although in many cases this was not because it is not technically viable.

In terms of NAT traversal no purely decentralized mechanism existed that worked with all NAT implementations. Nevertheless high success rates can be achieved when using a combination of them, although a third party is often indispensable. Because of the migration to the Internet Protocol version 6 the problem of being globally accessible via end-to-end can simplify middlebox traversal in a significant way. Because of that research is still needed in this area.

But since P2P is becoming more important every day, I cherish great expectations for the developement of decentralized secure peer-to-peer network applications.

7. REFERENCES

[1] J. Buford, H. Yu, and E. K. Lua. *P2P Networking and Applications*. Morgan Kaufmann, 2008.

[2] M. Conrad and H.-J. Hof. A generic, self-organizing, and distributed bootstrap service for peer-to-peer networks. *Self-Organizing Systems*, pages 59–72, 2007.

[3] C. Cramer, K. Kutzner, and T. Fuhrmann. Bootstrapping locality-aware P2P networks. In *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 1, pages 357–361. IEEE, 2004.

[4] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference, General Track*, pages 179–192, 2005.

[5] C. GauthierDickey and C. Grothoff. Bootstrapping of peer-to-peer networks. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on*, pages 205–208. IEEE, 2008.

[6] W. Ginolas. Aufbau eines virtuellen privaten Netzes mit Peer-to-Peer Technologie. Master's thesis, Fachhochschule Wendel, `http://p2pvpn.org/thesis.pdf`, 2009.

[7] P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. F. Riley, and E. Zegura. Bootstrapping in Gnutella: A measurement study. *Passive and active network measurement*, pages 22–32, 2004.

[8] M. Knoll, A. Wacker, G. Schiele, and T. Weis. Bootstrapping in peer-to-peer systems. In *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, pages 271–278. IEEE,

2008.

[9] M. Knoll, M. Helling, A. Wacker, S. Holzapfel, and T. Weis. Bootstrapping peer-to-peer systems using IRC. In *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09. 18th IEEE International Workshops on*, pages 122–127. IEEE, 2009.

[10] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Technical report, RFC 5766 (Proposed Standard), 2010.

[11] A. Müller, N. Evans, C. Grothoff, and S. Kamkar. Autonomous NAT traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010)*, IEEE, 2010.

[12] A. Müller. *Analysis and Control of Middleboxes in the Internet*. Techn. Univ. München, Lehrstuhl Netzarchitekturen und Netzdienste, 2013.

[13] n2n: a Layer Two Peer-to-Peer VPN. `http://luca.ntop.org/n2n.pdf`. `http://www.ntop.org/products/n2n`.

[14] J. Oikarinen and D. Reed. RFC 1459: Internet relay chat protocol. 1993.

[15] OpenVPN. `http://openvpn.net/`.

[16] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for NAT (STUN). Technical report, RFC 5389 (Proposed Standard), 2008.

[17] P. Saint-Andre. RFC 3920: Extensible messaging and presence protocol (XMPP). *Instant Messaging and Presence, IETFproposed Standard*, 2004.

[18] X. Shen *Handbook of peer-to-peer networking*. Springer, New York, 2010.

[19] I. Timmermans and G. Sliepen *tinc Manual*. `http://tinc-vpn.org/documentation-1.1/tinc.pdf`.

[20] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis. A NAT traversal mechanism for peer-to-peer networks. In *Peer-to-Peer Computing*, pages 81–83, 2008.

[21] D. I. Wolinsky, P. St Juste, P. O. Boykin, and R. Figueiredo. Addressing the P2P bootstrap problem for small overlay networks. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010)*, IEEE, 2010.

[22] D. I. Wolinsky, K. Lee, P. O. Boykin, and R. Figueiredo. On the design of autonomic, decentralized VPNs. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pages 1–10. IEEE, 2010.

[23] D. I. Wolinsky, L. Abraham, K. Lee, Y. Liu, J. Xu, P. O. Boykin, and R. Figueiredo. On the design and implementation of structured P2P VPNs. *arXiv preprint arXiv:1001.2575*, 2010.

doi: 10.2313/NET-2014-08-1_01