Analysis and Topology-based Traversal of Cascaded Large Scale NATs

Andreas Müller, Florian Wohlfart and Georg Carle
Chair for Network Architectures and Services
Technische Universität München
{mueller, wohlfart, carle}@net.in.tum.de

ABSTRACT

Middleboxes are an essential part of today's networks since they allow to introduce additional functionality without having to change end-hosts. Network Address Translation (NAT) has been the number one choice for coping with the address depletion problem of IPv4. Although NAT introduces many problems for existing applications it can be found in almost every consumer and mobile network.

Large Scale NAT (LSN) is the latest trend in middlebox deployment and plays an important role for the transition from IPv4 to IPv6. LSN may consist of a distributed NAT at the provider or it may include multiple layers of NAT. LSN introduces additional problems for customers since many existing NAT traversal techniques cannot be applied.

This paper presents an approach for discovering and measuring stateful cascaded NATs on the path between two arbitrary peers in the Internet. An algorithm combining multiple UDP packets, individual timeouts and traceroute measurements is presented and evaluated in a public field test. Finally, we show how NAT traversal for LSN can be improved by parameterizing existing algorithms according to the detected topology.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations

Keywords

Middlebox, NAT, LSN, Measurement, Field Test

1. INTRODUCTION

As the Internet was being designed and evolved, one of the most important design decisions was the end-to-end principle. It states, that application-specific functionality should be implemented in end-hosts only. With the growing size of the Internet, middleboxes have been introduced into the network to cope with emerging problems: NAT devices mitigate the Internet address depletion problem, caches and proxies help to efficiently distribute content, and firewalls protect networks from potential attackers. As middleboxes violate the end-to-end connectivity model of the Internet protocol suite they introduce numerous problems for services and applications. A large number of solutions to the so-called middlebox traversal problem have been developed [5, 11, 12], most of them focusing on the scenario where only one stateful middlebox on the path to the open Internet (e.g. a NAT at the customer's premises) exists.

One of the latest trends in middlebox deployment is Large Scale NAT (LSN) or Carrier Grade NAT (CGN). Instead of switching to IPv6, ISPs deploy multiple layers of middleboxes to serve their customers. This is not only true for mobile networks, as shown in [18, 17]: approaches such as Dual-Stack Lite [4] use LSN to deploy IPv6 in service provider networks without forcing the customers to switch to IPv6. With the deployment of LSN, many of the previously working NAT-Traversal techniques fail: control-based solutions, such as UPnP, are not applicable as they only control the innermost NAT, the provider's LSN cannot be reached. This not only has technical reasons (multicast), but also security reasons (lack of authentication). Behavior-based solutions, such as hole-punching, deliver suboptimal results as the TTL field of the hole-punching packet are difficult to set precisely due to the unknown topology [16]. Part of the problem is that it is not possible to detect and measure the exact number of NATs between two peers in the Internet, which would help to select an appropriate NAT traversal algorithm. Traditional NAT detection algorithms, such as STUN [12], only deliver the outermost IP address and treat multiple LSNs as one large black box. Thus, the detection of multiple cascaded NATs on the path between two arbitrary hosts is not possible without the hop-to-hop support of protocols such as NSIS/NSLP [15].

This paper proposes an algorithm that is capable of discovering previously unknown cascaded stateful middleboxes on the path from a client to a server located in the public Internet without having support of intermediate hops. Please note that in the rest of this paper we will often use the more generic term middlebox since our algorithm is not limited to NATs only. The algorithm is integrated in the NAT analyzing software NATAnalyzer[1] and run in many networks to discover the topology of ISPs and to gather information about the deployment of LSNs. Additionally, we show how the traversal of LSNs can be improved by parameterizing the well-known hole-punching algorithm based on the ac-

[1] http://nattest.net.in.tum.de

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HotMiddlebox'13, December 9, 2013, Santa Barbara, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2574-5/13/12 ...$15.00.
http://dx.doi.org/10.1145/2535828.2535833.

tual topology. This is especially helpful in scenarios where control-based techniques, such as port-forwarding or UPnP, are not applicable.

This paper is organized as follows: After giving an introduction to LSN in section 2, the state of the art in middlebox detection and related work in the context of LSN is presented in section 3. Section 4 then describes our approach in detail. Section 5 presents the results of our experimental analysis and section 6 concludes this paper.

2. LARGE SCALE NAT

Large Scale NAT (LSN) is an approach to shift NAT functionality from the customer to the ISP, especially in the context of the transition to IPv6. Some LSN deployments see address translation only at the provider's side, while others are based on cascaded and multi-layered NATs. The transition to IPv6 is difficult because a large number of hosts in private networks still only support IPv4 and many consumer electronic devices will never receive a firmware update supporting IPv6. Therefore, providing IPv6-only services to new customers is not possible. The second problem is that most content in the Internet is not yet reachable via IPv6. In fact, according to the Comcast's AAAA and IPv6 connectivity statistics[2] less than 5% of the top 1M websites provide an AAAA record. In this paper LSN refers to a setup that includes one or more NATs at the provider side with the option of having additional NATs deployed along the path, e.g. at the customer premises, resulting in an unknown topology of cascaded NATs.

3. RELATED WORK

Two different approaches for NAT discovery exist: The first one treats middleboxes as black boxes and assumes no direct control. For example, the STUN protocol [12] allows a peer to retrieve its IP address and port as seen by a STUN server to determine if it is located behind NAT. With STUN, the complete topology is treated as a black box and only if STUN servers were deployed in every intermediate network, it would be possible to reveal the complete topology by querying one after another.

The second discovery category defines interfaces and specifies protocols for querying and controlling middleboxes directly. The UPnP protocol is a typical example for this category, other proposals include the control extension for STUN [13], the "TCP Option for Transparent Middlebox Discovery" [9] and the Port Control Protocol (PCP) [19]. However, these extensions only work when in widespread use. Thus, the detection of multiple NATs and their position on the path is still an unsolved problem.

Evaluations of different hole-punching algorithms [2, 6] have shown that setting the TTL value for the initial hole-punching packet to a low value delivers better results because the initial packet does not reach the remote host. With only one NAT located usually only one hop from the end-host the determination of the TTL value is trivial. With an unknown topology, setting the TTL value to exactly the value that it traverses the outermost NAT without reaching the other host is rather difficult. In [16] the authors propose to use traceroute to count the number of hops and set the TTL value to half of the end-to-end hop count.

A number of experimental analysis and field tests covering network, middlebox and NAT behavior have been done in the past. In [5] a UDP hole-punching algorithm is evaluated using a small number of devices in the wild. TCP middlebox behavior is tested in [6]. [7] looks at home gateway characteristics such as TCP/UDP timeouts, DNS handling and ICMP messages. [8] tests basic UDP behavior, port mapping and filtering. Netalyzr [10] is a useful online tool to analyze the current network connection, e.g. in case of a network problem. The web-based test covers many networking aspects such as latency, but also a basic NAT detection test. The goal of the HomeNetProfiler [3] is to gather networking information of home routers to eventually improve the usability of home networks. NetPiculet [18] was designed to test firewall and NAT policies, such as timeouts, filtering and mappings. This was realized using a smartphone application and the test was run in a large number of cellular provider networks, thus also covering Large Scale NATs. [17] analyzes how provider-side LSNs allocate IP addresses based on the geolocation of the devices and [1] evaluates the impact of LSNs for residential broadband users in New Zealand.

4. DETECTION OF LARGE SCALE NAT

A naive approach to detect stateful middleboxes without requiring active support is to send out UDP messages with different TTLs. As soon as the packets expire, intermediate routers send ICMP TTL exceeded packets including the headers of the initial UDP packet. By comparing the IP addresses a client would be able to detect translating middleboxes. However, today's middleboxes also retranslate the headers in the ICMP payload, which makes this approach useless.

The initial assumption of our measurement algorithm is that we are able to exchange a sequence of packets between a client that wants to discover the topology and a server that is located in the public Internet. The server in the public Internet acts as a reference point that clients use to conduct their measurements as depicted in figure 1.

The first step for a client is to simply connect to the test server in order to establish a mapping in all intermediate nodes. Intermediate nodes may be legacy IP routers or one or more (stateful) middleboxes such as NATs, LSNs and firewalls. Once the server has received the first packet (e.g. a simple UDP packet), it counts the number of hops between the server and the client by conducting a traceroute[3] measurement. To get reliable results a stable path is required. As paths may vary due to load-balancing, the number of hops between the client and the server should be as low as possible. This can be reached by geographically distributing multiple servers in the public Internet.

In general, our approach does not require that NATs on the path decrement the TTL field as routers do. If a NAT does not decrement the TTL field, the next router will, so both hosts will be detected as one hop by the algorithm. The only inaccuracy that can arise from this issue is that two directly adjacent NATs could be deteced as one NAT instead of two separate NATs. Our goal is to find initial TTL values for low TTL hole-punching, so we can live with this limitation.

[2]http://www.employees.org/~dwing/aaaa-stats/

[3]http://linux.die.net/man/8/traceroute

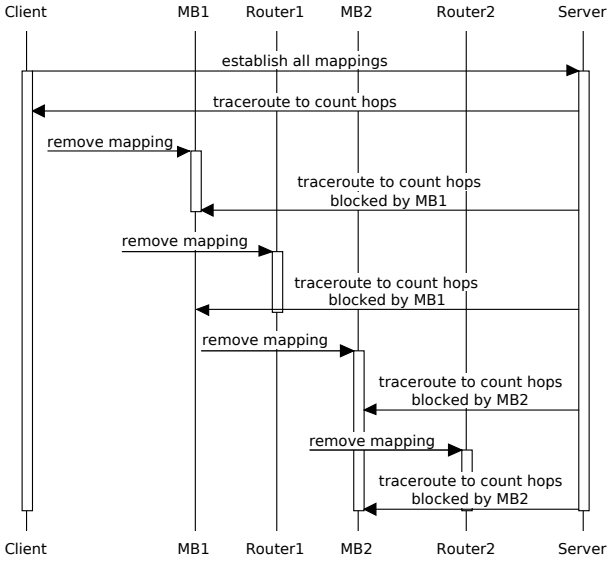

Figure 1: Topology detection approach

Once the server has detected the initial topology (the number of hops), the client successively removes each mapping starting from the one closest to it. After removing a mapping the server repeats the traceroute to the same destination and compares the number of hops with the initial result. In figure 1 the mapping of MB1 is removed first and the following traceroute from the server to the client is blocked by the middlebox since no state exists anymore. Thus, the result of the traceroute reveals one hop less than the initial hop count, which is used by the server as an indicator that there is a stateful middlebox at hop 1. In the second step the client directs Router1 to remove its mapping. Since it is no stateful middlebox the following traceroute still reaches MB1 with the same hop count as before. Thus, hop 2 does not implement stateful filtering. These steps are repeated for each hop, which eventually reveals the topology from the client toward the server. The pseudo-code of our algorithm is shown in the following:

```
establish mapping in all middleboxes along the path;
start from n=1, increment n by 1 for each run and
repeat
    remove mapping of all hops from 1 to n as seen by
    the client;
    count number of hops from server to client;
until n reaches number of hops;
```

Alg. 1: Middlebox topology detection algorithm

4.1 Remove Mapping

Our algorithm requires the capability of a client to remove a mapping in an intermediate node. Without a protocol that actively controls middleboxes this is not a trivial task. However, when considering stateful middleboxes, there are two possibilities to remove an established state: First, a *State Timer* is assigned to each mapping and once it expires, the mapping is deleted. Second, a *Policy* might exist that removes a mapping based on a special packet (e.g. a TCP-RST or TCP-FIN) or on a sequence of packets.

In case of TCP, a client would establish a regular TCP connection to the server, thus creating a mapping in each intermediate node. As long as the connection is in the established state, the mappings are kept in the middleboxes (assuming the TCP-established timer is large enough). As soon as a middlebox sees a TCP-RST packet it might (according to its actual behavior) remove the mapping immediately. However, if the client simply terminates the existing connection in the regular way, the TCP-RST packet is sent to the server via all intermediate nodes, which removes their mappings. Our algorithm calls for a strategy that requires to keep the TCP connection in the established state and only removes the mappings step by step. Thus, a TCP-RST packet has to be sent in a way that it only reaches a specific hop. This can be done by setting the TTL value of the IP packet to an appropriate value. As the Client's TCP-RST packets may need to pass other middleboxes with their mapping already removed, we need to presume that middleboxes always allow outbound traffic.

For UDP no such RST packet exists. Thus, waiting for a timeout is the only possibility to remove a mapping from a stateful middlebox. Unfortunately, timeouts of middleboxes differ, which requires to slowly approach the timeout experimentally. The following shows our algorithm for UDP:

```
foreach hop from 1 to n do do
    establish mapping: send UDP packet from client to
    server;
    server waits for UDP Timeout and sends keep-alive
    packets with TTL = #Hops − n;
    traceroute from server to client;
end foreach
```

Alg. 2: Middlebox topology detection algorithm for UDP

In order to make sure that only the timeout of the innermost middlebox (the one to be tested) expires, the server has to send keep-alive packets to all the other hops. This can be done by setting the TTL value of the corresponding IP packet in a way that the keep-alive packet expires right before it reaches the middlebox to be tested. Since the timeout value is not known beforehand and may vary from middlebox to middlebox, a large number of tests using different timeout values should be run in parallel.

The server keeps track of all traceroute measurements and stores the number of hops that were successfully traversed for each timeout value. For each such combination, the server also knows the value of the client's current counter (variable n in the above pseudo-code) representing the hop for which the client just removed the mapping. As a result, the server is able to derive the number of stateful middleboxes from it. A complete example on how to derive the position of stateful middleboxes is given in section 5 when presenting our experimental results.

4.2 Side Effects and Synergies

The topology measurements not only reveal information about stateful middleboxes on the path, it also delivers additional results. First, when using the UDP approach the timeouts of the individual hops are not known in advance, resulting in many parallel tests with increasing timeout val-

ues. When running a large number of tests, timeouts of each hop on the path can be detected. Second, port allocation patterns of the outermost middlebox can be detected by monitoring the source IP addresses and ports of packets coming from the client. Third, by comparing hop counts, the stability of a path can be examined. Finally, it can be observed if providers block outgoing ICMP messages, which is a real problem for getting accurate results. The following listing gives an overview of additional results that can be gathered by the algorithm:

- By running parallel tests with increasing timeouts, the individual timeout for the state can be detected for each hop.
- By monitoring source addresses of incoming packets, binding and port allocation patterns can be detected.
- By comparing hop counts, the path stability can be monitored. Unstable hop counts are strong indicators for load-balancing.
- By comparing TCP and UDP results, the filtering behavior for ICMP packets can be detected.

5. EXPERIMENTAL RESULTS

After having designed an algorithm to detect the topology of cascaded stateful middleboxes in section 4, we conducted a public field test to evaluate our approach. As mentioned, we integrated our topology detection test into the NAT-Analyzer software which combines a number of tests (e.g. STUN, UPnP, various hole-punching scenarios, etc.) to examine middlebox behavior in the wild. As the TCP-based version of our algorithm requires superuser privileges on the client side (which is not acceptable for most of the volunteers running the test), we only integrated the UDP-based version.

In total we collected the results of 4.810 tests, in which 93% completed the traditional middlebox analysis tests and 53% completed our new topology discovery test (which takes several minutes). We received test results from fixed-line providers using DSL, cable networks, mobile networks, LTE networks[4] and others such as a public WLAN access point and a satellite network. Apart from that, we received test results from organizations, which operate their own networks, such as larger companies and university campuses. In the following, we will refer to this type of connection as *corporate networks*. Most test results were submitted from Germany (19%) and the United States (11%), since we actively approached people to conduct our field test in these countries. In order to have short path lengths, we deployed two geographically distributed servers (United States and Germany) and used geolocation to distribute the clients.

There are two possibilities to detect multiple stateful middleboxes with our experimental results: First, by comparing IP addresses of the UPnP results with the actual public IP addresses and second by our topology detection algorithm as presented in section 4. Both possibilities were realized and their results are presented in the following.

5.1 UPnP Test Results

Our UPnP test is a client-side test that attempts to connect to a UPnP-enabled middlebox within the participant's network. If it succeeds, the test retrieves the device name, manufacturer and external IP address from the middlebox. Afterwards, these results are committed to our test server using HTTP-POST. By comparing the source IP address of this HTTP connection with the external IP address found in the UPnP test results, we have a first indicator for cascaded middleboxes. 26.5% of all UPnP-enabled devices carried an external UPnP address that was different from the one that was used to post the results. In 23.2% of these cases the UPnP address came from the 10/8 range, in 6.3% from the 172.16/16 range, in 63% from the 192.168/24 range and in 14% the external UPnP address was also a public one. Based on these UPnP test results, we can distinguish between cases with a *single NAT* (public external IP) and cases with cascaded NATs (private external IP). As we detected a significant number of setups where participants operate multiple cascaded NAT devices within their premises, we further divide the class of cascaded NATs into *Large Scale NAT* and *Multi NAT*. Multi NAT covers all cases where all NATs are operated within the customer's premises, while Large Scale NAT or LSN representes cases where at least one of the cascaded NATs is operated by the ISP.

The distinction between *Multi NAT* and *LSN* based on the UPnP results is fuzzy, so it was decided manually according to the following indicators for LSN: external IP addresses from the 10/8 range, with unusual high digits (e.g. 10.87.56.201), or whether other test results from the same provider network had similar addresses.

Finally, we had to deal with the results which showed a public external IP in the UPnP results, which did not match the IP address of the HTTP connection. In this case, a cascaded NAT scenario is unlikely since this would eclipse a part of the public IP address space from the user and make these addresses unreachable. Therefore, we assume that these results were caused by HTTP proxies or VPN tunnels and classify these groups as *Single NAT* results, with one exception: in some mobile and LTE networks, we discovered external IP addresses from the 100.64/10 address range, a space labeled as "Reserved Shared Address Space" by IANA.[5] These results are classified as *LSN*.

Table 1 summarizes the findings of our UPnP test using the three categories defined above.

Connection Type	UPnP Enabled	Single NAT	Multi NAT	LSN
Total	29.0%	76.6%	22.5%	0.9%
DSL	32.8%	73.3%	25.3%	1.4%
Cable	48.9%	91.3%	8.7%	0.0%
LTE	26.9%	42.9%	14.3%	42.9%
Mobile	5.0%	0.0%	33.3%	66.6%
Corporate	12.3%	93.8%	6.3%	0.0%

Table 1: UPnP test results

The results confirm our assumptions that UPnP-enabled middleboxes are prevalent in home networks connected via Cable (48%), DSL (32.8%) or LTE (26.9%). Furthermore, we found LSNs to be present in mobile networks (66.6%), LTE networks (42.9%) as well as fixed-line networks.

[4]"LTE connection" refers to LTE-based wireless internet access for homes in remote areas where fast fixed-line connections are not available.

[5]http://whois.arin.net/rest/net/NET-100-64-0-0-1

5.2 Topology Test Results

Our new approach for middlebox topology detection allows detecting individual middleboxes, even with cascaded stateful middleboxes on the path from the test client to the Internet. The UDP-based implementation requires that ICMP messages are generated and forwarded by intermediate nodes. However, we discovered that in many cases (48.4%) where stateful middleboxes are involved, the provider blocks outgoing ICMP messages as an answer to incoming UDP packets, which leads to problems with our algorithm. Therefore, for many results we were not able to reveal the exact topology. However, the proposed method is still valid for estimating the approximate position of the LSN and for determining a valid TTL value for hole-punching (which will be further described in the following section).

We found LSNs to be present in mobile networks, LTE networks, as well as fixed-line networks (e.g. the German provider EncoLine[6] most likely implements a Linux-based LSN), which confirms our earlier findings from the UPnP test.

In the following we show the topology of the Vodafone Germany LTE network that could be revealed using our test. The resulting table from our detection algorithm is shown in table 2.

	Hop 11	H 12	13	14	15	16	17(C)
20s	17	17	17	17	17	17	17
30s	16	16	16	16	16	16	17
50s	16	16	16	16	16	16	17
60s	16	16	16	16	16	16	17
90s	13	13	13	16	16	16	17
120s	13	13	13	16	16	16	17

Table 2: Result for the Vodafone Germany topology

For a timeout value of 20 seconds the test server is able to reach the client (hop 17) no matter which mapping the client tried to remove. Please note that removing a mapping for UDP means that the TTL value of the servers keep-alive packets is set to $n-1$, n being the number of hops as seen by the client. For a timeout of 30s the test server reaches hop 17 only if the mapping of hop 16 has not been removed. Thus, hop 16 implements a stateful middlebox with a timeout value between 20 and 30 seconds. The same is true for hop 13 that implements a stateful middlebox with a timeout between 60 and 90 seconds. The server is able to reach beyond hop 13 only if either the timeout has not triggered yet (here 60s) or if the mapping has not been removed yet (columns hop 14 to hop 17). With this table we are able to generate the corresponding topology as shown in figure 2.

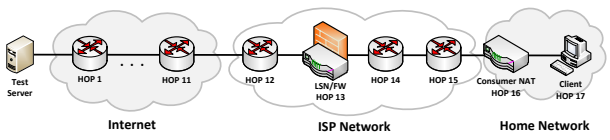

Figure 2: Revealed topology for Vodafone LTE

[6]http://www.encoline.de/

Analogous to the classification of the UPnP results we also developed a classification scheme for the topology test results:

- *Blocked* includes all cases where the outgoing ICMP traffic was blocked by the provider, so we could not reveal the NAT topology.
- *No NAT* means each traceroute got through to the end-host. This can have two different reasons: either there is no stateful middlebox along the path or it could not be detected due to a large UDP timeout value.
- *Single NAT* contains results where exactly one NAT could be detected.
- *Cascaded NAT* means two or more NATs could be detected.

In addition, 24% of the results could not be classified by our algorithm and remain as *unknown*. Table 3 summarizes the findings of our topology test, which revealed 48.4% blocked, 14.7% no NAT, 12.5% single NAT and 0.4% cascaded NAT connections.

Connection Type	Blocked	No NAT	Single NAT	Cascaded NAT
Total	48.4%	14.7%	12.5%	0.4%
DSL	38.0%	17.6%	15.4%	0.0%
Cable	52.5%	15.5%	13.0%	0.0%
LTE	75.0%	0.0%	8.4%	16.7%
Mobile	70.5%	6.8%	2.3%	18.2%
Corporate	44.8%	22.4%	19.0%	0.0%

Table 3: Topology Results from our field test.

Using the topology test results we can now identify the optimal TTL value for hole-punching packets and evaluate the LSN hole-punching solution developed in [16]. Therefore, we determine the hop distance between the end-host and the outermost NAT device in our results. In cases where our topology test was not blocked, we found that this distance was 5 hops or lower.

As a large share of our tests (48.4%) was blocked and did not allow us to find the hop distance directly, we could at least estimate the hop distance in these cases. For that purpose we use the hop distance from the end-host to the hop that blocked outgoing ICMP packets as an upper bound for the end-host to outermost NAT distance. We can estimate this distance via the reverse path length, which we infer from the TTL values of incoming UDP packets. Despite the tentativeness of this approach, it was able to show that the number of hops is 5 or lower in 89.0% of the tests.

These findings show that an outgoing packet with an initial TTL value of 5 or above will traverse all stateful middleboxes and reach the public Internet. Then we looked at typical path lengths between two peers to make sure packets don't reach an opposite NAT. In our test, we found that 97.0% of all paths had 10 or more hops. As we deliberately kept our paths short for our test, we can conclude that an initial TTL value of 5 allows low TTL hole-punching in 89.0% of all cases. This value works in nearly all cases, so we can state that it is not necessary to use our topology detection algorithm or the approach from [16] to set the initial TTL value.

5.3 Implications for Traversal

When combining both the results of our NAT behavior tests (STUN, UPnP, hole-punching, etc.) with the results from our topology test, we can draw further conclusions regarding the traversal of the tested middleboxes. Table 4 shows the NAT types as defined in the STUN protocol specification [14] in order of increasing restrictiveness: Full Cone, (Port) Address Restricted and Symmetric NAT grouped by the observed NAT classes. One can observe an increased restrictiveness from Single NAT over Multi NAT to LSN, which means LSN scenarios are on average more difficult to traverse than Multi NAT scenarios.

Connection	Full Cone	Port Rest.	Sym-metric	Avg. UDP To.
Total	13.3%	50.3%	17.4%	42.7s
Single NAT	10.9%	64.0%	6.1%	48.6s
Multi NAT	3.0%	49.3%	27.4%	38.3s
LSN	7.1%	64.3%	28.6%	47.6s

Table 4: Observed NAT Types

Table 5 compares hole-punching success rates for the observed NAT classes. It compares standard hole-punching to low TTL hole-punching, an adapted version of hole-punching where the punching-packets are sent with a custom inital TTL value to make sure the packet does not reach the opposite NAT. Two interesting findings can be observed in table 5: first, UDP low TTL hole-punching performs worse than UDP high TTL hole-punching and second, TCP low TTL hole-punching performs significantly better than TCP high TTL hole-punching, especially for LSNs. This shows the relevance of low TTL hole-punching and our effort of finding an optimal TTL value.

	UDP HP		TCP HP	
Connection	Std.	Low TTL	Std.	Low TTL
Total	73.5%	56.0%	38.1%	40.3%
Single NAT	87.7%	65.5%	57.8%	58.7%
Multi NAT	61.8%	45.9%	23.6%	31.4%
LSN	57.7%	50.0%	21.4%	50.0%

Table 5: Measured hole-punching success rates

6. CONCLUSION

Large Scale NATs and other stateful middleboxes are already deployed by many providers and cause additional problems for middlebox traversal. In this paper we present an algorithm to identify the topology of cascaded NATs on the path between two arbitrary peers in the Internet. We propose two ways of using a combination of packets to create mappings, remove mappings and traceroutes to determine the individual positions. Afterwards, the algorithm was evaluated in a public field test. As its two main results we could show that many ISPs already deploy LSNs and that the success rate for the traversal of NAT in general depends on the network topology. Since the field test showed that the current implementation of the algorithm delivers inaccurate results in cases where intermediate nodes implement specific filtering rules, we propose an initial TTL value that should be used for hole-punching, especially for TCP.

Future work is to examine the TCP-based algorithm as also proposed in this paper, as well as new prediction algorithms in case of blocked ICMP packets.

7. REFERENCES

[1] S. Alcock et al. Investigating the impact of service provider NAT on residential broadband users. In *Infocom*, San Diego, CA, USA, March 2010.

[2] A. Biggadike et al. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *ACM SIGCOMM Asia Workshop*, 2005.

[3] L. DiCioccio et al. Probe and Pray: Using UPnP for Home Network Measurements. In *PAM conference*, Vienna, Austria, March 2012.

[4] A. Durand et al. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. RFC 6333, Aug. 2011.

[5] B. Ford et al. P2P communication across NAT. In *USENIX*, Anaheim, CA, USA, 2005.

[6] S. Guha and P. Francis. Characterization and measurement of TCP traversal through NATs and firewalls. In *IMC*, 2005.

[7] S. Hätönen et al. An Experimental Study of Home Gateway Characteristics. In *IMC*, Melbourne, Australia, November 2010.

[8] C. Jennings. NAT Classification Test Results. Internet Draft - expired, IETF, July 2007.

[9] A. Knutsen and A. Ramaiah. TCP option for transparent Middlebox discovery. Internet Draft - work in progress, IETF, February 2012.

[10] C. Kreibich et al. Netalyzr: Illuminating The Edge Network. In *IMC*, Melbourne, Australia, Nov. 2010.

[11] J. Rosenberg. Interactive Connectivity Establishment (ICE). RFC 5245, Apr. 2010.

[12] J. Rosenberg et al. Session Traversal Utilities for NAT (STUN). RFC 5389, Oct. 2008.

[13] J. Rosenberg and H. Tschofenig. Discovering, Querying, and Controlling Firewalls and NATs. Internet Draft - expired, IETF, October 2007.

[14] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, Mar. 2003.

[15] M. Stiemerling et al. NAT/Firewall NSIS Signaling Layer Protocol (NSLP). RFC 5973, Oct. 2010.

[16] K. Tobe et al. Extended UDP Multiple Hole Punching Method to Traverse Large Scale NAT. In *Asia Pacific Advanced Network Meeting*, Hanoi, Vietnam, August 2010.

[17] S. Triukose et al. Geolocating IP addresses in cellular data networks. In *PAM*, Vienna, Austria, 2012.

[18] Z. Wang et al. An untold story of middleboxes in cellular networks. In *ACM SIGCOMM 2011*, Toronto, ON, Canada, 2011.

[19] D. Wing et al. Port Control Protocol (PCP). RFC 6887, Apr. 2013.