Autonomous NAT Traversal

Andreas Müller Nathan Evans Christian Grothoff
Network Architectures and Services
Technische Universität München
Email: {mueller,evans,grothoff}@net.in.tum.de

Samy Kamkar
Email: samy@samy.pl

***Abstract*—Traditional NAT traversal methods require the help of a third party for signalling. This paper investigates a new autonomous method for establishing connections to peers behind NAT. The proposed method for autonomous NAT traversal uses fake ICMP messages to initially contact the NATed peer. This paper presents how the method is supposed to work in theory, discusses some possible variations, introduces various concrete implementations of the proposed approach and evaluates empirical results of a measurement study designed to evaluate the efficacy of the idea in practice.**

I. Introduction

A large fraction of the hosts in a typical peer-to-peer network are in home networks. Most home networks use network address translation (NAT) [1] to facilitate multiple computers sharing a single global public IP address, to enhance security or simply because the provider's hardware often defaults to this configuration. Recent studies have reported that up to 70% of users access P2P networks from behind a NAT system [2]. This creates a well-known problem for peer-to-peer networks since it is not trivial to initiate a connection to a peer behind NAT. For this paper, we will use the term *server* to refer to a peer behind NAT and the term *client* for any other peer trying to initiate a connection to the server.

Unless configured otherwise (protocols such as the Internet Gateway Device Protocol [3] are counted as configuration in this context), almost all NAT implementations refuse to forward inbound traffic that does not correspond to a recent matching outbound request. This is not primarily an implementation issue: if there are multiple hosts in the private network, the NAT is likely unable to tell which host is the intended recipient. Configuration of the NAT is not always an alternative; problems range from end-user convenience and capabilities of the specific NAT implementation to administrative policies that may prohibit changes to the NAT configuration (for example, due to security concerns).

Since NAT systems prohibit inbound requests that do not match a previous outbound request, all existing NAT traversal techniques (aside from those changing the configuration of the NAT system) that we are aware of require some amount of active facilitation by a third party [4], [5]. The basic approach in most of these cases is that the server in the private network behind the NAT is notified by the third party that the client would like to establish a connection. The server then initiates the connection to the client. This requires that the server maintains a connection to a third party, that the client is able to locate the responsible third party and that the third party acts according to a specific protocol.

The goal of this paper is autonomous NAT traversal, meaning NAT traversal without a third party. Using third parties increases the complexity of the software and potentially introduces new vulnerabilities. For example, if anonymizing peer-to-peer networks (such as GNUnet [6] or Tor [7]) used third parties for NAT traversal, an attacker may be able to monitor connections or even traffic volumes of peers behind NATs which in turn might enable deanonymization attacks [8], [9]. Another problem is that the decrease in available globally routable IPv4 addresses [10] will in the near future sharply reduce the fraction of hosts that would be able to facilitate NAT traversal.

II. Technical Approach

The proposed technique assumes that the client has somehow learned the current external (globally routable) IP address of the server's NAT. This could be due to a previous connection between the two systems or a third party having provided the IP address in a previous exchange. Note that we specifically assume that no third party is available at the time when the client attempts to connect to the server behind the NAT.

The first goal of the presented NAT traversal method is to communicate the public IP address of a client that wants to connect to the server behind the NAT. After the server is aware of the IP address of the client, it connects to the client (similar to NAT traversal methods that involve a third party).

The key idea for enabling the server to learn the client's IP address is for the server to periodically send a message to a fixed, known IP address. The simplest approach uses ICMP ECHO REQUEST messages to an unallocated IP address, such as 1.2.3.4. Since 1.2.3.4 is not allocated, the ICMP REQUEST will will not be routed by routers without a default route; ICMP DESTINATION UNREACHABLE messages that may be created by those routers can just be ignored by the server.

As a result of the messages sent to 1.2.3.4, the NAT will enable routing of replies in response to this request. The connecting client will then fake such a reply. Specifically, the client will transmit an ICMP message indicating `TTL_EXPIRED` (Figure 1). Such a message could legitimately be transmitted by any Internet router and the sender address would not be expected to match the server's target IP.

The server listens for (fake) ICMP replies and upon receipt initiates a connection to the sender IP specified in the ICMP

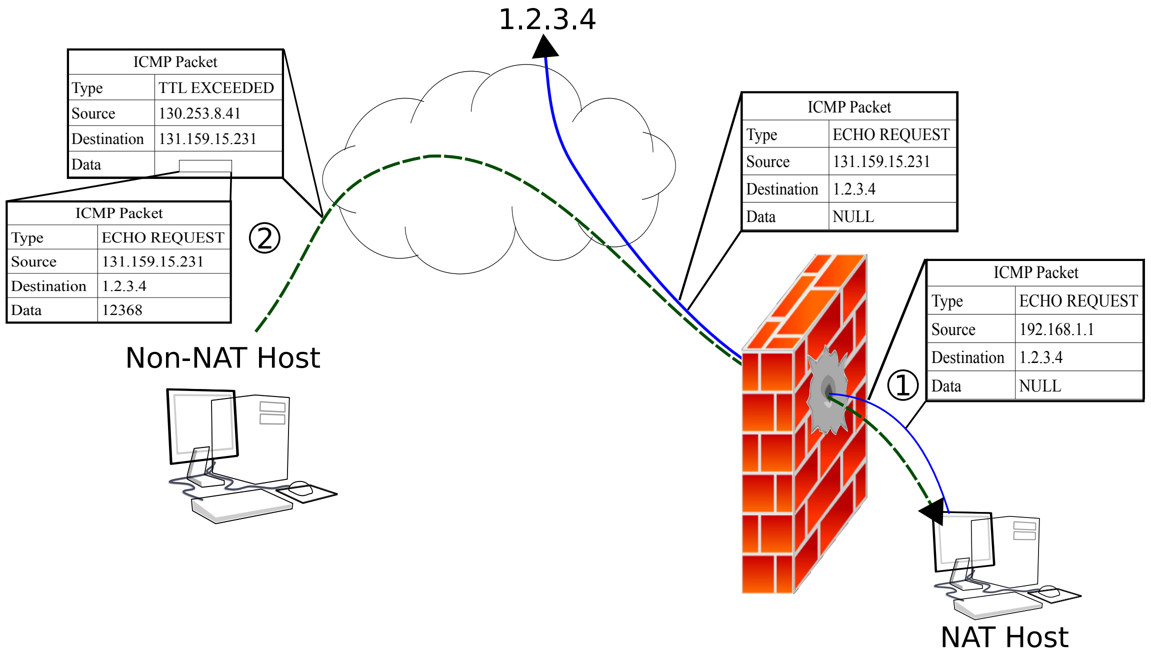

Fig. 1. This figure diagrams the process of sending and receiving the fake ICMP messages for the server and client. In step 1, the server sends a fake ICMP request to 1.2.3.4 and in step 2 the client sends the matching reply. Note that this is a fake reply since the client *never* receives the ICMP request sent to 1.2.3.4 by the server. The important information contained in the actual packets is displayed for each step. The blue (solid) line shows the ICMP request path and the dashed (green) line shows the ICMP reply path.

reply. If the client is using a globally routable IP address, this is entirely unproblematic and both TCP or UDP can be used to establish a bi-directional connection if the client listens on a pre-agreed port. In cases where there is no pre-agreed port, a port number can in most cases be communicated as part of the payload of the ICMP ECHO RESPONSE, which is typically not checked against the payload of the corresponding ICMP ECHO REQUEST by NAT implementations.

A. NAT-to-NAT Communication

Further complications arise if both the client and the server are behind NAT. In this case, often the client will be unable to transmit a fake ICMP response to the server due to restrictions imposed by the NAT implementation of the client. One possible idea for circumventing this problem is for the client to send the same message that the server is sending except with TTL 1 to its NAT. If the NAT accepts the packet despite the forged sender IP address it might theoretically generate the desired ICMP response and forward it to the external network. However, in practice we did not find NATs where generating the necessary ICMP message using a TTL of 1 works.

Even if the client is able to transmit the fake ICMP response, the next step; in which both the client and server are aware of the others IP address and now intend to establish a TCP or UDP connection can still be complicated. The reason is that NAT systems can change the source port numbers of outbound messages. Without a third party, both client and server would have to guess matching source and destination port numbers as chosen (possibly at random) by their respective NAT implementations. Depending on the type of the NAT implementations (Full cone, restricted cone, port-restricted, symmetric), finding the correct port may take several messages. Client and server can reduce the total number of messages required by transmitting and listening on multiple ports in this phase.

B. Using UDP packets instead of ICMP ECHO REQUESTs

A possible alternative to having the sender transmit ICMP ECHO REQUESTs to a fixed, known IP address is having the sender transmit UDP packets to a fixed, known IP address and port. In this case, the client would again forge an ICMP TTL_EXPIRED message, only this time using the UDP format. The main disadvantage of this variation is that the sender has to guess the external UDP sender port number when faking the ICMP response. Since some NAT implementations randomly change those port numbers, the server might have to send UDP packets using multiple sender ports in order to give the client a sufficient chance at guessing correctly.

The main advantage of this technique is that the server no longer needs to send using RAW sockets, which may reduce the privileges required for the server. Note that the server still needs to be able to listen for the ICMP reply, which requires RAW sockets on Linux. In the case of a full-cone NAT, using UDP packets instead of ICMP ECHO REQUESTs also has the advantage of establishing a port mapping which can then be used as an alternative method for contacting the peer.

Another difference between the two approaches is the possible payload that can be embedded in the response. With ICMP ECHO REQUESTs, the payload can be as big as the packet size permits and is hence only limited by the MTU of the respective physical network. Well-formed ICMP UDP TTL exceeded replies on the other hand can only contain 32 bits of payload: the ICMP TTL_EXCEEDED response contains the first 64 bits of the payload of the original IP packet. In those 64 bits, the 16-bit UDP checksum field and the 16-bit UDP

TABLE I

EXPERIMENTAL EVALUATION OF AUTONOMOUS NAT TRAVERSAL. "ECHO-SERVER" LISTS THE NUMBER OF NAT IMPLEMENTATIONS THAT ALLOWS (FAKED) ICMP TTL_EXPIRED REPLIES TO TRAVERSE THE NAT IN RESPONSE TO ICMP ECHO REQUEST MESSAGES. "ECHO-CLIENT" LISTS THE NUMBER OF NAT IMPLEMENTATIONS THAT ALLOW CLIENTS TO TRANSMIT (FAKED) ICMP TTL_EXPIRED MESSAGES. "UDP-SERVER" AND "ICMP-UDP-CLIENT" GIVE THE SAME NUMBERS WHEN USING UDP PACKETS INSTEAD OF ICMP ECHO REQUESTS. "PRESERVES PORTS" INDICATES THE NUMBER OF IMPLEMENTATIONS THAT PRESERVE THE SENDER'S LOCAL PORT AS THE EXTERNAL PORT IF POSSIBLE. "ANY SERVER" LISTS THE NUMBER OF NATS WHERE EITHER THE ECHO-SERVER OR THE UDP-SERVER WORK. FINALLY, "TWO-MESSAGE SUCCESS" LISTS THE NUMBER OF NATS WHERE AUTONOMOUS NAT TRAVERSAL (AS A SERVER) SUCCEEDS EITHER WITH ECHO-SERVER OR WITH UDP WITH PORT PRESERVATION AND HENCE ONLY TWO MESSAGES ARE NECESSARY TO REACH THE SERVER.

	Echo-Server	Echo-Client	UDP-Server	ICMP-UDP-Client	Preserves Ports	Any server	Two-Message Success
Full cone	0/4	1/4	1/4	1/4	0/4	1/4	0/4
Restricted cone	9/31	5/34	26/40	5/34	16/43	26/40	9/31
Port-restricted	37/56	2/71	82/91	2/71	72/98	83/91	43/56
Symmetric	2/3	2/5	3/5	2/5	6/6	3/5	2/3
Overall	53/103 (51%)	10/123 (8%)	121/149 (81%)	10/123 (9%)	100/162 (62%)	122/149 (82%)	62/103 (60%)

packet length are unverifiable (for NAT's that do not track extensive information about outgoing UDP packets) and can hence be used to transmit 32 bits of information to the server (in addition to the sender's IP address). With our approach, either of these payload sizes is enough as we only transmit a port number in addition to the IP address.

III. IMPLEMENTATIONS

This section summarizes the three implementations of the proposed method that we have done so far. All of the presented implementations are freely available from the web pages of the respective projects.

A. Implementation in NAT-Tester Framework

Our implementation in the NAT-Tester framework was used to gather the data for this paper. It transmits the various packet types (with or without payload) using raw sockets and uses `libpcap` to determine which messages were forwarded by the NAT. The client is currently available for W32 and Linux and must be run with administrator rights. This implementation is useful for researchers interested in exploring the various variations of this and other NAT traversal methods.

B. Implementation in `pwnat` tool

The `pwnat` tool[1] is a GNU/Linux-only stand-alone implementation of autonomous NAT traversal. After contacting the server behind the NAT, it establishes a channel with TCP-semantics using UDP packets. It supports both client and server behind NAT (if one of the NATs allows the fake ICMP messages to be transmitted). This implementation targets end-users.

C. Implementation in the GNUnet Framework

Finally, we have created a re-usable implementation of the presented ICMP-based NAT traversal method in GNUnet, GNU's framework for secure peer-to-peer networking [6]. Since the use of ICMP requires the use of non-portable and often privileged system calls, this implementation is split into three main components:

ICMP server
: This component is a small program that provides the core ICMP-related functionality for the server. The code periodically generates the ICMP ECHO REQUEST message and also listens for incoming ICMP TTL_EXCEEDED responses. If such a response is received, it simply prints the IP address of the sender to `stdout`. If the ICMP also contains a 16-byte payload, it is interpreted as a port number and also printed.

ICMP client
: This component is a small binary which simply sends a single (fake) ICMP message to the IP address specified at the command line. An additional argument can be given which will be interpreted as a port number to be transmitted in the payload of the fake ICMP response message.

Transport plugin
: This component implements a GNUnet transport plugin [11] and is thus specific to the GNUnet peer-to-peer framework. Depending on how the peer is configured, it controls ICMP servers or clients and ultimately establishes connections between peers.

Splitting the implementation into these three components has the advantage of minimizing the amount of code that must run with super-user privileges on POSIX systems (by installing the ICMP server and client with the SUID bit set). Furthermore, since the ICMP code is platform-specific, this makes it easier to manage this platform-specific part of the code. Finally, this split makes it easy to share the platform-specific but peer-to-peer network agnostic ICMP code so that it can be used with other peer-to-peer applications. This implementation is suitable as a starting point for developers of P2P networks.

IV. EXPERIMENTAL RESULTS

We have evaluated the proposed autonomous NAT traversal techniques on a large number of NAT implementations using our NAT-Tester framework [12], [13]. The framework consists of a public client that volunteers download and execute. The client then performs various tests against the local NAT implementation and reports the results back to the NAT-Tester server. This enables us to evaluate NAT traversal strategies against a wide range of NAT implementations. Detailed results are made public on the NAT-Tester web page.[2] In this section

[1]http://samy.pl/pwnat/

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

we will summarize the results based on the data available so far.

Table I summarizes which fractions of the NAT implementations evaluated so far support the proposed method for autonomous NAT traversal. We distinguish between behavior relevant for using autonomous NAT traversal from the point of view of both clients and servers behind NAT. We consider two cases: the case where the server uses ICMP ECHO REQUESTs and the case where the server transmits UDP packets. We also consider the extend of UDP port randomization which determines how efficient the second stage in the case of NAT-to-NAT communication would be. NAT implementations are categorized into the typical four types (full cone, restricted cone, port-restricted, symmetric) in cases where NAT-Tester is able to determine the type. NAT implementations that do not seem to fall into any of these categories are only included in the total.

The data shows that in virtually all cases NATs forward the faked ICMP messages for UDP (UDP-Server), but only in about half the cases for ICMP ECHO REQUESTs (Echo-Server). Furthermore, a significant majority of all NATs also preserve the source port (when possible), so the additional requirement of guessing the port for faking the ICMP response for a UDP message does not change the overall cost of the approach. Finally, NATs virtually always prevent their clients from transmitting the fake ICMP messages used by our clients (Echo-Client, ICMP-UDP-Client). Based on what we have seen from inspecting NAT configurations directly, the reason seems to be that NAT rules typically only allow ICMP packets for the states "NEW" and "ESTABLISHED" in the state machine [14] — and the fake response falls into neither category.

V. Discussion

The proposed method of autonomous NAT traversal works well in the case of an unrestricted client attempting to initiate a connection to a server behind NAT. Here, in virtually all cases a single ICMP message by the client would be followed by traditional connection reversal [15] which then reliably creates a UDP or TCP connection. In other words, there is no need for third parties to help initiate connections to NATed servers in this case.

On the other hand, if both systems are behind NAT, the proposed method rarely works and a third party is required. Assuming 70% of the peers in a network are behind NAT, this means that roughly 50% of all possible connections can be established using autonomous NAT traversal. However, even in the case where both systems are behind NAT a possible advantage of the proposed method remains; it is easy to create a simple, generic and fully stateless service that receives requests from NATed peers and generates fake ICMP replies to notify the server behind NAT. In this case, the payload of the ICMP reply would need to contain the original IP address (and likely source port number) of the client since the IP header of the faked ICMP response would now contain the IP address of the service.

VI. Conclusion

Fake replies can enable autonomous NAT traversal in a number of cases. As with most NAT traversal techniques, this approach does not work for all installations. What is unusual about the presented method is that it works extremely well if only one peer is behind NAT and virtually never if both peers are behind NAT. Systems that require high NAT traversal success rates typically implement a number of traversal techniques and the presented approach extends the set of available methods by one that, if applicable, is cheaper and simpler than most of the existing techniques.

Acknowledgment

The authors thank the many volunteers that have downloaded and executed the NAT-Tester and helped us gather data to evaluate the method. This work was funded in part by the Deutsche Forschungsgemeinschaft (DFG) under ENP GR 3688/1-1.

References

[1] K. Egevang and P. Francis, "Rfc 1631: The ip network address translator (nat)," 1994.

[2] M. Casado and M. J. Freedman, "Illuminating the shadows: Opportunistic network and web measurment," December 2006. [Online]. Available: http://illuminati.coralcdn.org/stats/

[3] P. Iyer and U. Warrier, *InterngetGatewayDevice:1 Device Template Version 1.01*, UPnP Forum, http://www.upnp.org/standardizeddcps/igd.asp, November 2001.

[4] J. Rosenberg and R. Mahy et. al., "Session Traversal Utilities for NAT (STUN)," RFC 5389, IETF, October 2008.

[5] J. Rosenberg, R. Mahy, and P. Matthews, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," RFC 5766 (Review Copy), IETF, February 2010.

[6] K. Bennett and C. Grothoff, "gap - Practical Anonymous Networking," in *Designing Privacy Enhancing Technologies*. Springer-Verlag, 2003, pp. 141–160.

[7] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004. [Online]. Available: citeseer.ist.psu.edu/dingledine04tor.html

[8] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, May 2005, pp. 183–195.

[9] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on tor using long paths," in *18th USENIX Security Symposium*. USENIX, 2009, pp. 33–50.

[10] G. Huston, "Ipv4 address report," March 2010. [Online]. Available: http://www.potaroo.net/tools/ipv4/

[11] R. A. Ferreira, C. Grothoff, and P. Ruth, "A Transport Layer Abstraction for Peer-to-Peer Networks," in *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (GRID 2003)*. IEEE Computer Society, 2003, pp. 398–403.

[12] A. Müller, A. Klenk, and G. Carle, "On the Applicability of knowledge-based NAT-Traversal for future Home Networks," in *IFIP Networking 2008, Springer, Singapore*, May 2008.

[13] A. Müller and A. Klenk and G. Carle, "Behavior and Classification of NAT devices and implications for NAT-Traversal," *IEEE Special issue on Middleboxes*, pp. 14–19, September 2008.

[14] G. N. Purdy, *Linux iptables Pocket Reference*. O'Reilly Media, Inc., 2004.

[15] P. Srisuresh, B. Ford, and D. Kegel, "Rfc 5128: State of peer-to-peer (p2p) communication across network address translators (nats)," 2008.