Digital Object Identifier: 10.1109/NCA.2011.66

URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6038609

SYNI – TCP Hole Punching Based on SYN Injection

Sebastian Holzapfel, Matthäus Wander, Arno Wacker, Torben Weis

University of Duisburg-Essen, Distributed Systems Group,
Bismarckstraße 90, 47057 Duisburg, Germany
{*sebastian.holzapfel*|*matthaeus.wander*|*arno.wacker*|*torben.weis*}*@uni-due.de*

***Abstract*—The shortage of IPv4 addresses and the very slow transition to IPv6 leads to pragmatic solutions in the Internet: today many hosts are still using IPv4 and are connected to the Internet over a Network Address Translation (NAT) router. However, there are many applications, which need inbound connections, like e.g. peer-to-peer-based systems or voice-over-IP. For such NATed hosts inbound connections usually pose a problem, since without additional measures the router filters the incoming connection attempts. These additional measures are usually referred to as NAT traversal mechanisms and *hole punching* is one of those techniques. In this paper we propose a new protocol for a TCP-based hole punching mechanism based on self-injecting SYN-packets in the local network stack.**

I. Introduction

Network Address Translation (NAT) is a well-established pragmatic solution to overcome the shortage of IPv4 addresses in the Internet [1]. According to [2], [3] it is currently widely used with more than 70% of all hosts accessing the Internet being situated behind a NAT router. With IPv6 there would be enough IP addresses for every possible host, but the transition to IPv6 is proceeding slowly and therefore NAT will remain an established fact for the coming years.

With NAT a single public IP can be shared among multiple hosts behind a NAT router. The hosts behind such a router use private IP addresses, which are replaced with the public address of the router as the packets pass. All corresponding responses are therefore addressed to the public address. Upon arrival of such a response the NAT router replaces the public address with the private internal address and thus allows the hosts in the private network a transparent Internet connectivity. From the Internet only the public IP address of the NAT router is directly reachable and thus hosts behind a NAT router cannot be addressed directly. This poses a problem for services relying on inbound connections, e.g. peer-to-peer-based (P2P-based) systems or voice-over-IP (VoIP). Therefore, so-called NAT traversal mechanisms are required to allow inbound connections for such services.

One NAT traversal mechanism is *Universal Plug and Play* (UPnP) which is simple to use but deactivated on most NAT routers by default [4]. If UPnP is not available, *hole punching* can be used. Hereby the host behind a NAT router starts by sending a packet to an external endpoint in the Internet. By doing so the NAT router creates a mapping and forwards from then on incoming packets corresponding to this mapping to the internal host. Due to the connectionless nature of UDP this technique is in general simpler to perform with UDP than with TCP. However, many applications require a reliable data transport. Even though there exist protocols like RUDP [5] which provide a reliable transport based on UDP, in most cases using the approved and thoroughly investigated TCP is the better choice.

Our goal is a TCP hole punching mechanisms that succeeds with high probability. To achieve this goal, it is necessary to rely on properties which are met by hosts and NAT routers in practice. Avoiding unusual communication flows that may be blocked by some NAT routers and adhering as far as possible to the TCP specification increases the probability of a successful hole punching attempt. Additionally the approach shall be robust in practical environments, i.e. do not rely on harsh timing constraints that are hard to control in practice.

In this paper we present our new protocol SYNI for TCP NAT traversal using hole punching with self-injected SYN-packets. The main idea of our approach is to additionally transport the opening SYN packet of a TCP connection over an out-of-band communication channel from the source to the destination host and inject it locally at the destination host. This SYN packet could have never reached the destination directly due to the NAT router in front of the destination host.

II. System Model

In this section we define the system model used throughout the paper. We assume that computers (*hosts*) are connected to the Internet using NAT routers, possibly behind multiple levels of NAT routers. These routers translate and filter incoming and outgoing packets according to a rule set. Hosts are able to determine the external IP address of the outer router as well as the assigned external port number. This can be done by using the STUNT protocol defined in [6] or the MFB protocol defined in [4].

Hole punching requires that hosts are able to exchange information over a common communication channel. This may be e.g. a well-known rendezvous server which is directly reachable without hole punching. In serverless peer-to-peer systems directly reachable superpeers may be used to

pass messages between hosts that are not yet interconnected. Independent whether servers or superpeers are used, this communication channel is slower than a direct connection between hosts and can be a bottleneck when too many messages are exchanged. Therefore, it should only be used to negotiate the connection establishment between hosts.

III. Fundamentals

In this section we discuss some challenges occurring when two hosts behind different NAT routers attempt to use hole punching to establish a TCP connection. When the NAT assigns an external port to an internal IP endpoint, the NAT router is able to forward all packets that are sent to this external port. If a NAT router receives packets on an external port that is not assigned to an internal IP endpoint, these packets will be dropped. These packets can be dropped silently or the NAT router can respond with a message. The response can be an ICMP (*Internet Control Message Protocol*) message with type *Destination Unreachable* or a TCP message with the RST (reset) flag set. A TCP RST is typically treated as fatal error and will probably cause the socket to behave differently than when a timeout occurs. The error response may also cause the NAT router to remove the mapping, which cancels any hole punching attempt. To avoid receiving such a message, most of the TCP hole punching mechanisms (see Section VI) need to ensure that an outgoing message passes the local NAT routers to create a mapping, but does not arrive at the remote NAT router. To do so, the TTL (*Time To Live*) value of the IP header has to be reduced so that the message will be dropped before it reaches its destination. The applicable TTL value (*low TTL*) depends on the number of hops between the source and destination. Usually, a host is behind one NAT router and thus a TTL of 2 will be sufficient to pass the local NAT router but not arrive at the destination NAT router. In case of multi-level NAT with different routes the low TTL needs to be determined for each destination individually before the hole punching attempt begins. This can be done by sending IP packets, e.g. with UDP or TCP payload, repeatedly to the destination with incremented TTL values until an *ICMP Destination Unreachable* or TCP RST message is received. A message that does not reach its destination because of an expired TTL will lead to an *ICMP Time Exceeded* message. While it depends on the operating system and the NAT router being used, an *ICMP Time Exceeded* is in general less likely to be fatal for the connection attempt than a TCP RST.

NAT routers maintain a dynamic mapping table to track the states of the TCP connections being forwarded. This may be a rudimentary tracking of outgoing TCP messages or a more sophisticated tracking combined with stateful packet inspection for packet filtering purposes. Thereby, the NAT router filters incoming and outgoing messages that are contrary to the current TCP state. For example, each TCP connection is established by a three way handshake. After sending an outgoing TCP message with the SYN flag set, the NAT router expects an incoming message with the SYN and ACK flags set. If then the NAT router receives an incoming message with the SYN flag but not the ACK flag set, the message contradicts the TCP state and may be dropped. Similar to missing mappings, the NAT router can filter incoming packets silently, respond with an ICMP error or respond with a TCP RST message. This applies as well to outgoing messages, e.g. the NAT router can forward an outgoing SYN-ACK that follows an outgoing SYN or it could reject this message because it is contrary to the current TCP state.

IV. SYNI Approach

We now describe a new TCP hole punching mechanism that is based on the injection of self-created TCP SYN messages. Host A, located behind NAT router R_A, wants to establish a connection to a host B which is located behind NAT router R_B.

Our approach consists of four steps as shown in Figure 1. It has some prerequisites to the operating systems of the hosts A and B and the involved NAT routers R_A and R_B. We number the prerequisites **(Px)**, since we use them in Section VI to compare different hole punching mechanisms.

In the first step A creates a plain TCP stream socket and sets a low TTL value. When initiating the socket connection to B's external IP endpoint, A uses a raw socket to capture all outgoing messages on its network device **(P1)**. By using the raw socket A retrieves the initial sequence number (ISN) of the outgoing SYN message, under the prerequisite that R_A does not rewrite it **(P2)**. Some NAT routers support adding a random offset on the ISN to protect against IP spoofing. It is however uncommon for home devices – in our test described in [4] we did not encounter a NAT router altering the sequence numbers. When the SYN message passes R_A, R_A creates a mapping for A's connection request and forwards the SYN message onward on the route to B. Because of the low TTL value however the SYN message does not arrive at R_B and thus R_B will not respond with a TCP RST or a similar fatal error, that could result in a socket error at A. Instead, R_A receives an *ICMP Time Exceeded* message from any router along the path between R_A and R_B. Independently whether R_A forwards the ICMP message back to A or not, R_A shall keep the mapping open and A shall keep the connecting socket open waiting for a SYN-ACK response **(P3)**. A sends a message over the common communication channel to B containing the external IP endpoint of A as well as the the ISN of the captured SYN message.

In the second step, when B has received the connection request from A, it creates a raw socket that is able to send TCP messages. B sets up a SYN message with B's internal IP endpoint as source, a low TTL value, a random sequence number and A's external IP endpoint as destination. Then,

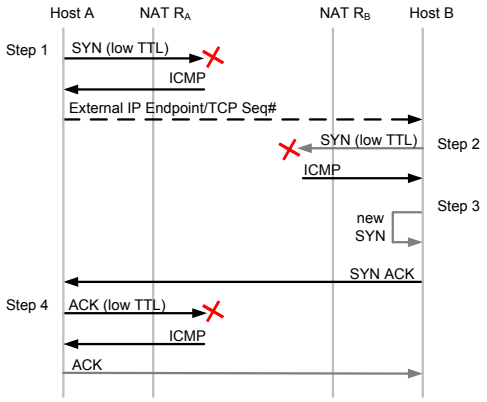

Figure 1. SYNI - dotted lines are messages exchanged over a common communication channel

B uses the raw socket to send this message via R_B which forwards the message to the Internet. Because of the low TTL value the message will never arrive at R_A or A but R_B will have set up a mapping to forward packets from B to R_A. Similar to step 1, R_B will receive an *ICMP Time Exceeded* message which shall not affect the newly created mapping on R_B **(P3)**.

The third step is about creating and injecting a SYN message at B. B uses the information received over the common communication channel with A to create a SYN message that looks as if it has been sent by A. In the TCP header, B sets A's external port as source port, the ISN as sequence number and B's internal listening port as destination port. As IP header, B sets A's external IP address as source and B's internal IP address as destination. The TCP/IP packet is encapsulated in a link layer protocol, which is usually Ethernet. In the Ethernet frame B sets its hardware address as destination and any reasonable source address, e.g. R_B's hardware address. B uses the raw socket created in the previous step to inject the SYN message into B's TCP/IP stack and then closes the raw socket. The self-injected SYN message will not be sent over the network because it is addressed to B. As B has a listening socket associated to the destination IP endpoint addressed in the SYN message, B's TCP/IP stack creates a connection state and responds with the corresponding SYN-ACK message. B's operating system sends the SYN-ACK message over the network to R_B which already has a mapping for this packet flow and forwards the SYN-ACK to R_A.

In the fourth step R_A forwards the SYN-ACK message to A, because from R_A's perspective this SYN-ACK message corresponds to the previously sent SYN. A receives this SYN-ACK message which matches the connection state of the outgoing connection to B and thus the TCP/IP stack will send a corresponding ACK message. A needs to reset the TTL value of the connecting socket to the default value to ensure that the ACK message arrives at the destination. Due to practical limitations it is not possible to change the TTL on all operating systems within the TCP handshake, i.e. after the SYN out but before the ACK out (Winsock error on Windows 7). Thus A resets the TTL value once the socket is in connected state which is after A's TCP/IP stack has responded with an ACK message with a low TTL value. A captures the outgoing ACK message as it did with the SYN message in step 1 and re-sends it by using the raw socket with the default TTL value. Though two ACK messages are forwarded by R_A, only the one with the default TTL value will arrive at R_B.

After the ACK message is transmitted via R_B to B, the connection is established and both A and B have interconnected plain TCP stream sockets to exchange data. From the perspective of R_A, the TCP connection was established by a normal three way handshake, except that a duplicate ACK message has been sent which is in accordance with the TCP specification. From R_B's perspective, the connection is established with the unusual sequence SYN out, SYN-ACK out, ACK in **(P4)** which is forwarded by most NAT routers as we will show in the next section.

V. Evaluation

To evaluate our approach, we built a test setup with 12 different NAT routers in the lab (Table I). 10 NAT routers are embedded devices targeting the small office/home office or consumer market segment and 2 NAT routers are operating system distributions running on PC hardware. The WAN ports of the NAT routers are connected to a Linux PPPoE server which simulates the Internet Service Provider and routes between the devices under test. A UDP rendezvous server on another host helps to determine the external IP endpoint and serves as common communication channel. The LAN ports of the NAT routers are connected to two test machines, one acting as host A (SYNI connection initiator) and one as host B (SYNI listener). We reset all NAT routers to factory defaults respectively installation defaults and did not change the configuration apart from the IP subnets and PPPoE logins.

As we observed during our testing, some NAT routers do not decrease the TTL value of forwarded packets properly. To ensure that packets with a TTL value of 2 pass away in transit before being forwarded to the destination NAT router, we use the Linux Netfilter on the PPPoE server to decrement the TTL value of each packet by 3, forcing an *ICMP Time Exceeded* message. In practice there are several hops between R_A and R_B, so that packets will definitely time out in transit. We implemented an automated test software using .NET C# and WinPcap [7]. WinPcap is necessary under Windows, because since Windows XP SP2 it is not possible to send or self-inject TCP messages via standard raw sockets. The two test machines are running Windows XP SP3 and Windows 7, but we also verified that the self-injection mechanism works on Linux as well.

Brand	Model	Firmware Version
Asus	RX3041	Runtime Code: V2.1.2.114 Boot Code: V0.1.5.24
AVM	FRITZ!Box 7050	14.04.33
Digitus	DN-11004-O	02.00.00.03
D-Link	DI-604 D1	V3.02
IPCop	Linux distribution	1.4.20
Level One	FBR-1415TX	R1.94p0v
Netgear	FM114P	Version 1.4 Release 26
Netgear	RP614 v3	V6.0GR
Netgear	FVS318	v3.0_26RC1
pfSense	FreeBSD distribution	1.2.3
SMC	7004ABR	Runtime Code: 1.00 Boot Code: V0.15T
SMC	7004VBR	Runtime Code: R1.00 Boot Code: R1.0606.0707

Table I
NAT ROUTERS TESTED IN THE LAB

With three NAT routers (pfSense, Netgear RP614, Asus RX3041) we could not determine the external IP endpoints reliably. One might use sophisticated port prediction techniques to try to determine the IP endpoints, but this is out of scope of this paper. We thus skipped these routers and tested all possible combinations of the remaining 9 NAT routers, resulting in $9 \cdot 8 = 72$ test cases. One test case consists of the following procedure on both hosts:

1) Set up default route to a NAT router
2) Determine own external IP endpoint with the help of the rendezvous server
3) Run SYNI, either as initiator or listener
4) Exchange payload over stream socket to ensure that the connection has been established or not

56 out of the 72 test cases completed successfully (78%). Taking the NAT routers into account for which the hosts could not determine the external IP endpoints automatically, this corresponds to 56 out of 132 cases (42%). After analysis of the network traces we confirmed that all of the successful tests performed SYNI as expected. 16 test cases timed out and were unsuccessful. In all of the failed tests IPCop or the Level One FBR-1415TX have been on the SYNI listener side (R_B). IPCop does not forward an unsolicited SYN-ACK out after sending a SYN out, thus not meeting **P4**. Similar to this, the FBR-1415TX does allow SYN out, SYN-ACK out, but does not allow an ACK in afterwards, thus also not meeting **P4**. As long as not both hosts are behind one of these routers, A and B can still establish a connection by switching the roles of the initator and listener.

In further experiments we discovered that the FBR-1415TX does allow the sequence SYN-ACK out, ACK in without any preceding SYN out, i.e. by skipping step 2 of SYNI. However, two other NAT routers (Netgear FM114P, Netgear FVS318) fail then, resulting in an overall lower success probability. While testing and debugging we also found special characteristics of certain NAT routers which allow for other hole punching attempts. For example, the Digitus

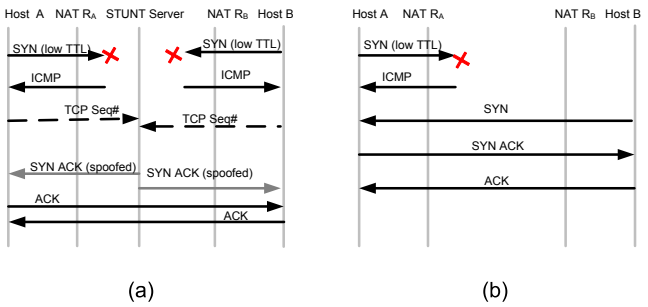

Figure 2. NUTSS [9] - dotted lines are messages sent to STUNT server

DN-11004-O does not allow the sequence SYN out, SYN in but it does allow the sequence SYN out, RST in, SYN in. The TCP RST, usually a fatal error, resets the connection state without removing the mapping, thus allowing a simple TCP connect to the Digitus' external IP endpoint. Hence, based on these special characteristics, new hole punching mechanisms can be developed.

In addition to the experiments in the lab we also tested SYNI between two hosts connected to the Internet via two different broadband Internet Service Providers. One host was located behind a *T-Com SpeedPort W701V* router and the other behind an *AVM FRITZ!Box 7140* router. We used the network diagnosis tool *traceroute* to determine the *low TTL* value used by the initiator. The initiator and listener determined their external IP endpoints by using the MFB protocol [4] and connected to a rendezvous server. SYNI worked in both directions successfully. We also tested whether the SYN message in step 2 was necessary in this setup: it was not necessary when the FRITZ!Box 7140 was the listener, but it was necessary when the W701V was the listener, as this router does not allow an unsolicited SYN-ACK out.

VI. Related Work

In this section we describe related TCP hole punching approaches and compare their prerequisites with SYNI. There are NAT traversal mechanisms based on UDP [3], [8], however, as we specifically target TCP, we will not go into the details of those approaches. All TCP hole punching mechanisms presented in the following require that A and B are able to inform each other about the intent of establishing a connection and to exchange the external IP endpoints, e.g. by using a common communication channel. We summarized the list of additional prerequisites in Table II.

A. NUTSS

Guha et al. presented in [10] an architecture called NUTSS that includes two different TCP hole punching mechanisms. The first approach as shown in Figure 2 (a) utilizes a so-called STUNT server that is used to send spoofed TCP messages. Both hosts A and B simultaneously

	P1		P2		P3		P4		P5	P6		P7		P8	
	A	B	A	B	A	B	A	B		A	B	A	B	A	B
NUTSS (a)	✓	✓	✓	✓	✓	✓	-	-	✓	-	-	-	-	-	-
NUTSS (b)	-	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-
P2PNAT	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓
NATBLASTER	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-
SYNI	✓	✓	✓	✓	✓	✓	-	✓	-	-	-	-	-	-	-

P1 RAW Socket
P2 No TCP sequence number rewriting
P3 Unaffected by *ICMP Time Exceeded*
P4 SYN out, SYN-ACK out, ACK in
P5 Spoofing
P6 No TCP RST when socket closed
P7 Socket option SO_REUSEADDR
P8 SYN out, SYN in

Table II
PREREQUISITES OF THE PRESENTED TCP HOLE PUNCHING MECHANISMS

initiate the TCP handshake by connecting to each other with a plain TCP stream socket. They use a raw socket **(P1)** to capture the ISN of the outgoing packet and then send this information to the STUNT server. This requires that both NAT routers R_A and R_B do not rewrite TCP sequence numbers **(P2)** or the approach will fail. The hosts use a low TTL value to ensure that an outgoing SYN message passes the local NAT router but does not arrive at the remote end, where it could trigger a TCP RST. Both NAT routers receive an *ICMP Time Exceeded* message from any router along the path between A and B. Hence, NUTSS (a) requires for a successful connection attempt that these ICMP messages do not interfere with the TCP mapping states of the NAT routers **(P3)**. When the STUNT server receives the ISNs, it sends spoofed SYN-ACK messages to A and B that seem to originate from each other. The need of spoofing TCP messages **(P5)** is a significant drawback since many Internet Service Providers perform *egress filtering* to prevent such messages from passing their network. If the STUNT server is egress filtered, the approach cannot be used.

The second approach [10] (see Figure 2 (b)) does not rely on spoofing TCP messages to set up a TCP connection. A initiates a connection to B using a low TTL value with a plain TCP stream socket. The SYN message never arrives at R_B, but R_A creates a new mapping and then receives an *ICMP Time Exceeded* message. Thus, R_A is required to not modify the mapping after receiving an *ICMP Time Exceeded* message **(P3)**. Shortly after initiating the connection, A closes the connecting socket and creates a new listening socket using the same port as used before. This approach requires that the connecting socket does not send a TCP RST message when it is closed **(P6)** because TCP RST is a fatal error and may impact the TCP state of the mapping on R_A. B then initiates a connection to A's listening socket. The connection will be established if R_A allows the unusual sequence of packets SYN out, SYN in **(P8)**.

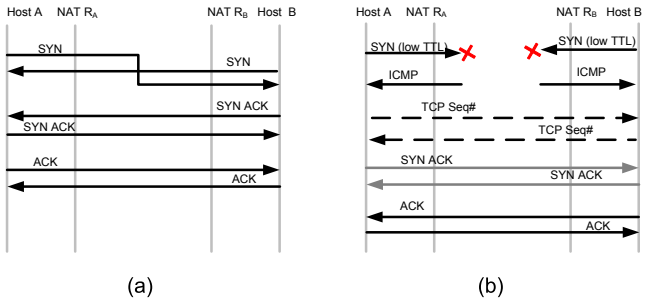

Figure 3. P2PNAT and NATBLASTER [9] - dotted lines are messages exchanged over a common communication channel

B. P2PNAT

Ford et al. presented in [8] a TCP NAT traversal mechanism (P2PNAT) that is based on the TCP simultaneous open scenario defined in [11]. A and B listen for an incoming connection on a port that will also be used to initiate an outgoing connection. Typically, a port number is used by a single socket for either incoming or outgoing connections. To bind multiple sockets to the same local port, the hosts needs to support the socket option *SO_REUSEADDR* **(P7)**. Both hosts initiate an outgoing connection at the same time to each other as shown in Figure 3 (a). Sending the outgoing SYN messages creates a mapping on R_A and R_B. If A and B manage to time the connection attempts in a way that both mappings are created before the remote SYN messages arrive, R_B and R_A can forward them to the destinations B and A. This requires that both NAT routers allow the sequence SYN out, SYN in **(P8)**. Afterwards the corresponding SYN-ACK as well as the ACK messages are sent and the connection is established. It is necessary that both hosts initiate the connection at the same time, because otherwise the remote NAT router can respond with an *ICMP Destination Unreachable* or a TCP RST message. This timing dependency makes P2PNAT highly sensitive to latency and jitter, thus showing a behavior that is difficult to control in practice. Depending whether the hosts involved

implement simultaneous open as specified in TCP, especially in combination with *SO_REUSEADDR*, it can also lead to different socket behaviors, requiring a complex handling in the implementation and being not very robust.

C. NATBLASTER

In [12] the authors present a TCP hole punching mechanism called NATBLASTER. This approach is similar to the NUTSS approach shown in Figure 2 (a) but does not send spoofed TCP messages. A and B initiate a connection using a low TTL value as shown in Figure 3 (b). Both hosts capture the ISN of the outgoing SYN message by using a raw socket **(P1)** and exchange it over a common communication channel. This requires R_A and R_B to not rewrite sequence numbers **(P2)**. R_A and R_B are also required to not being influenced by the *ICMP Time Exceeded* messages that both routers receive **(P3)**. Unlike in P2PNAT, the SYN messages do not arrive at their destinations and thus A's and B's TCP/IP stacks will not respond. Instead, A and B use the ISN to create a corresponding SYN-ACK message manually and send it to each other by using a raw socket. Upon receipt, A's and B's TCP/IP stacks respond with ACK messages and the connection is established.

The prerequisites of SYNI are similar to those of NATBLASTER. Both approaches require to send an outgoing SYN-ACK message without a corresponding incoming SYN message, and then to receive an incoming ACK message **(P4)**. The difference, though, is that NATBLASTER requires this for both NAT routers whereas SYNI requires this only for R_B. As explained in Section V, we have determined two NAT routers for which **P4** does not apply and where NATBLASTER would fail.

VII. Conclusion & Future Work

In this paper we presented a new TCP hole punching mechanism based on the injection of self-created TCP SYN messages. As a proof of concept we implemented our mechanism, tested it successfully in the lab and in the Internet. We compared the prerequisites of our approach with similar approaches and conclude that it has reasonable chances to succeed in practical scenarios in which other approaches do not work. With respect to the used routers our approach uses only one deviation from the TCP protocol specification (i.e. the destination router sees an outgoing SYN-ACK without an incoming SYN). The approach consists of few steps and does not require exceedingly tight timings, thus being well-suited for robust implementations.

We experienced in our experiments that NAT routers show very different behavior, especially when dealing with unusual packet flows. We conclude that network trace analyses are essential when evaluating hole punching mechanisms to differ between an effective mechanism and a lucky coincidence. To further increase the probability of successful TCP hole punching, hosts need to identify the behavior of their NAT routers and choose the best suitable hole punching mechanism subject to the information determined. It is therefore useful to have a set of different hole punching mechanisms which have different prerequisites and succeed in different scenarios.

References

[1] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," IETF, RFC 3022, 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt

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

[3] A. Mueller, N. S. Evans, C. Grothoff, and S. Kamkar, "Autonomous NAT Traversal," in *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P'10)*, IEEE. Delft, The Netherlands: IEEE, 2010. [Online]. Available: http://grothoff.org/christian/pwnat.pdf

[4] S. Holzapfel, M. Wander, A. Wacker, L. Schwittmann, and T. Weis, "A new protocol to determine the nat characteristics of a host," in *Proceedings of 25th IEEE International Parallel Distributed Processing Symposium, International Workshop on Hot Topics in Peer-to-Peer Systems (HOTP2P)*, Anchorage, Alaska, USA, May 2011.

[5] T. Bova and T. Krivoruchka, "Reliable UDP protocol," Internet Engineering Task Force, Internet-Draft, 1999. [Online]. Available: http://tools.ietf.org/id/draft-ietf-sigtran-reliable-udp-00.txt

[6] S. Guha, "STUNT – Simple Traversal of UDP Through NATs and TCP too," Network Working Group, Tech. Rep., December 2004, work in progress. [Online]. Available: http://nutss.gforge.cis.cornell.edu/pub/draft-guha-STUNT-00.txt

[7] CACE Technologies, "WinPcap," published in the WWW at http://www.winpcap.org, CACE Technologies. [Online]. Available: http://www.winpcap.org

[8] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *In USENIX Annual Technical Conference*, 2005, pp. 179–192.

[9] S. Guha and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[10] S. Guha, Y. Takeda, and P. Francis, "NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity," in *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. New York, NY, USA: ACM, 2004, pp. 43–48.

[11] J. Postel, "Transmission Control Protocol," IETF, RFC 793, September 1981. [Online]. Available: http://tools.ietf.org/html/rfc793

[12] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, "NATBLASTER: Establishing TCP connections between hosts behind NATs," in *Proceedings of ACM SIGCOMM ASIA Workshop*, Apr. 2005.