A B A K Ó S
PUC Minas Instituto de Ciências Exatas e Informática

Beware: NAT Traversal is a Simple and Efficient Approach to Open Firewall Holes[*]

Cuidado: Travessia de NAT é uma Abordagem Simples e Eficiente Para Abrir Buracos em Firewall

Elias P. Duarte Jr.[1]
Kleber V. Cardoso[2]
Micael O. M. C. de Mello[3]
João G. G. Borges[4]

Resumo

As técnicas de travessia de NAT permitem que processos com endereços IP privados não roteáveis se comuniquem com outros processos fora dos limites de segurança da rede. Técnicas como UDP Hole Punching foram padronizadas pelo IETF e, usando túneis baseados nessas técnicas, é fácil permitir que processos de aplicativos em cima de qualquer protocolo de transporte, incluindo TCP, iniciem e recebam pacotes da Internet através de dispositivos NAT. No entanto, como efeito colateral, essas técnicas também passam livremente por firewalls. Neste trabalho, descrevemos como é possível configurar qualquer servidor rodando em qualquer porta (sem nenhuma configuração especial de firewall) para estabelecer conexões iniciadas em clientes de Internet arbitrários, tornando serviços não autorizados facilmente disponíveis. Mostramos também que o processo é leve, principalmente depois de concluída a configuração inicial, suportando virtualmente qualquer tipo de aplicativo não autorizado.

Palavras-chave: Tradução de endereço da rede. Travessia de NAT. Perfuração (de firewall). Travessia de firewall. Transparência de rede. Falhas de segurança.

[*]submetido em 05/03/2019 e aceito em 07/08/2020.

[1]Professor do Departamento de Informática (DInf) da Universidade Federal do Paraná (UFPR), Brasil– elias@inf.ufpr.br

[2]Professor do Instituto de Informática (INF) da Universidade Federal de Goiás (UFG), Brasil– kleber@inf.ufg.br

[3]Mestre pelo Instituto de Informática (INF) da Universidade Federal de Goiás (UFG), Brasil– micael-mello@gmail.com

[4]Mestre pelo Departamento de Informática (DInf) da Universidade Federal do Paraná (UFPR), Brasil– borgesjg@gmail.com

Abstract

NAT traversal techniques allow processes with private, non-routable IP addresses to communicate with other processes outside the network secured limits. Techniques such as UDP Hole Punching have been standardized by the IETF, and using tunnels based on those techniques it is easy to allow application processes on top of any transport protocol, including TCP, to both start and receive packets from the Internet across NAT devices. However, as a side effect those techniques also freely proceed through firewalls. In this work we describe how it is possible to configure any server running on any port (no firewall configuration required) to establish connections initiated at arbitrary Internet clients, making unauthorized services easily available. We also show that the process is lightweight, in particular after the initial setup is concluded, thus virtually supporting any type of unauthorized applications.

Keywords: Network address translation. NAT traversal. Hole punching. Firewall traversal. Network transparency. Security holes.

1 INTRODUCTION

Transparency and end-to-end connectivity are two of the basic design principles of the original Internet (GARRETT et al., 2018; SALTZER et al., 1984). Informally, these concepts refer to the ability to communicate from any host to any other host in the network. The current Internet employs multiple techniques – including NAT (Network Address Translation) and – that have a direct impact on Internet transparency. According to Carpenter (CARPENTER, 2000) "the loss of transparency in the Internet is both a bug and a feature from the security viewpoint". Although these techniques are meant ensure security, in this work we show that they can be actually cause security breaches.

NAT traversal techniques (HUITEMA, 2006; HO et al., 2011; TSENG et al., 2013) were first developed due to the fact that several protocols, such as SIP (Session Initiation Protocol), are simply not able to traverse NAT devices. Those techniques are also required by multiple P2P applications as well as VoIP and videoconferencing which involve the communication of processes running on hosts at different administrative domains that are nearly always protected by NAT/firewall. Processes with private, non-routable IP addresses can use NAT traversal to freely communicate with other processes outside the network secured limits. Recently NAT traversal techniques have been developed in the context of the Internet of Things, to allow access to mobile devices (SRIRAMA; LIYANAGE, 2014) and also distribute software updates (HERRY et al., 2018). NAT traversal has also been applied in cloud computing (HSU et al., 2018), to allow clients and servers to remain connected even after migrations.

NAT traversal services are widely available, e.g. (PANASONIC, 2013; PRO, 2013; WEBRTC, 2013). NAT traversal techniques for UDP flows have been standardized by the IETF, including ICE (Interactive Connectivity Establishment) (ROSENBERG, 2010). ICE is based on STUN (Session Traversal Utilities for NAT) (ROSENBERG et al., 2008) and its extension TURN (Traversal Using Relay NAT) (MAHY et al., 2010). STUN implements UDP hole punching (SRISURESH et al., 2008) which allows UDP packets to flow between hosts behind NAT. Both (FORD et al., 2005) and (MÜLLER et al., 2013) confirm that most networks connected to the Internet feature NAT devices that support UDP hole punching.

In this work we discuss the fact that, as a side effect, the communication with NAT traversal techniques causes security breaches by freely opening holes on firewalls. Stateful firewalls employ the same principle as NAT for allowing packet flows, i.e. as the firewall processes the first packet of a given flow originated from an internal host it creates state information that allows the destination host to send back packets to the source host in the protected network. By using NAT traversal any internal host can independently open holes on any port without any firewall configuration to provide unauthorized services that can be accessed from the rest of the Internet.

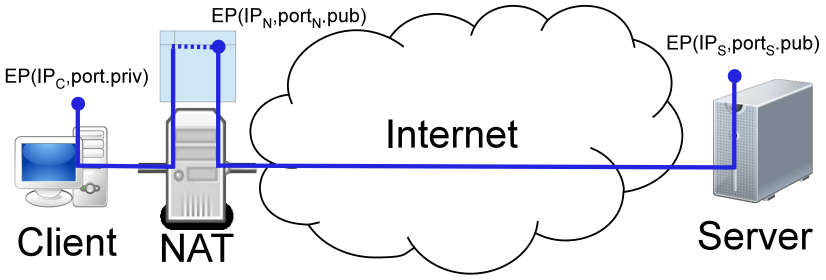
We describe a simple NAT traversal approach to allow application processes using any transport protocol to communicate seamlessly accross NAT devices and firewalls using IP-over-UDP tunnels. Note that using this technique it is possible to configure any server running on any port to establish connections initiated at arbitrary Internet clients. The system creates a virtual IPv6 tunneling interface at each participant host. As an application process sends a packet using the local virtual interface, the packet is encapsulated in a UDP datagram and sent to the destination using UDP hole punching. The destination then receives and injects the packet on the local virtual interface so that it is processed by the transport protocol as if it had come from a physical interface. We report experimental results that show that this approach is very efficient, in particular after tunnel setup is completed. As the overhead is very low, it can be employed by any type of application.

In the next section we first give a brief overview of NAT and firewalls and describe NAT traversal techniques. Next, in Section 3 NAT traversal strategy that allows communications with any transport protocol is presented. Experimental evaluation results are given in Section 4. Finally Section 5 is the conclusion.

2 NAT & FIREWALL TRAVERSAL

A NAT (Network Address Translation) device (SRISURESH; EGEVANG, 2001) connects two networks by translating IP addresses and TCP/UDP ports as they are forwarded. By using NAT, a set of hosts with private addresses can communicate on the Internet using a single public address. Figure 1 shows a client on a private network sending a UDP packet to a UDP server on the Internet. The client creates the packet unaware that the IP address is non-routable; the NAT device between the private network and the Internet makes the required address and port translations before the packet is forwarded. When the first packet is sent by the client, it is captured by the NAT device, which creates a local NAT session with enough information to execute the following steps for every packet sent to/from the pair of nodes: (1) translate and map the source private IP address of the packet to the public NAT IP address; (2) translate and map the UDP source port to an external port N which is used in the identification of the NAT session.

Figure 1 – A UDP session is established between a client behind NAT and a server



Source: Elaborated by authors.

When the server sends back a response to the client, the NAT device performs the reverse mapping and translation using the local NAT session information, i.e. it (1) reverses the public address to the private IP address of the client, (2) reverses the port back to the original one. Each process communicating through the NAT device has an associated endpoint (EP) consisting of the pair ⟨IP address, port⟩. The UDP session initiated by client A in Figure 1 has three EPs: the private EP used by the client; the public EP mapped by the NAT for the UDP session; and the public EP at the server. The sessions at the NAT device are dynamically maintained at a NAT session table. A session is removed from the table when no packet of that session has been seen by the NAT device for a predefined timeout interval. A NAT device only forwards a packet from the Internet to the private network if there is a NAT session that matches the packet information already configured at the session table. Thus NAT also acts as a stateful packet filter, in the sense that it prevents or allows packets to enter the private network.

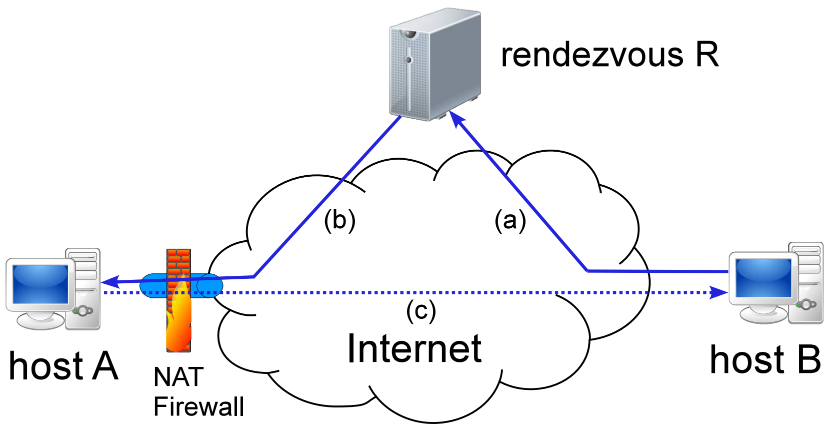
Firewalls (FREED, 2000; BODEI et al., 2018) are filters that inspect all traffic passing between two networks. A firewall is often placed at the gateway of a private network to the Internet. A firewall can block or allow packets to pass according to a set of rules. Stateful firewalls maintain information about the network and transport layers, some also about the activities of the application layer. Using a stateful firewall a packet originated at an external host in the Internet can enter the private network only if previously a packet was sent from the private network to the external host.

NAT traversal strategies can be roughly classified in two types: solutions that employ relay servers and those that employ the connection reversal technique. The second type is the most popular and efficient and it is the basis of the technique which we describe in this work. The first type employs a relay server that is not restricted by NAT nor firewall. Hosts behind NAT/-firewall establish communication with the relay server, which acts as a communication relay. The IETF has proposed a standard to traverse NAT using a relay server, TURN (MAHY et al., 2010). Despite its reliability, solutions that involve relays do have disadvantages (SRISURESH et al., 2008). In particular, the relay server is the least efficient approach for providing communication between hosts behind NAT/firewall. This approach depends on both the relay server processing resources and network bandwidth. Furthermore, even if resources are not a problem, the communication latency between the end-hosts is likely to increase significantly.

The connection reversal technique, shown in Figure 2, is simple and is the basis for approaches more efficient than the relay. In general, there is a rendezvous server and at least one host is not blocked by NAT/firewall. The rendezvous server does not relay traffic between end-hosts, it only takes part of the initial steps intermediating the establishment of a direct UDP flow between the hosts themselves. A host behind NAT/firewall needs initially to communicate with a rendezvous server in order to receive requests from outside. As it is shown in Figure 2, if **host B** wants to communicate with **host A**, it **(a)** sends a reverse connection request to the **rendezvous R**. The request **(b)** is relayed to **host A**, containing information about **host B**. Finally, **(c) host A** starts the reverse connection to the **host B**. The main drawback of this solution is the quite strong assumption that one of the end-hosts is not behind NAT/firewall.

There are several solutions for allowing the communication between two hosts both behind NAT/firewall (HUITEMA, 2006; ROSENBERG, 2010). These solutions are based on the same principles as those employed when only one host is behind NAT/firewall. The most successful approach (SRISURESH et al., 2008) is UDP hole punching. Before this is described

Figure 2 – Connection reversal



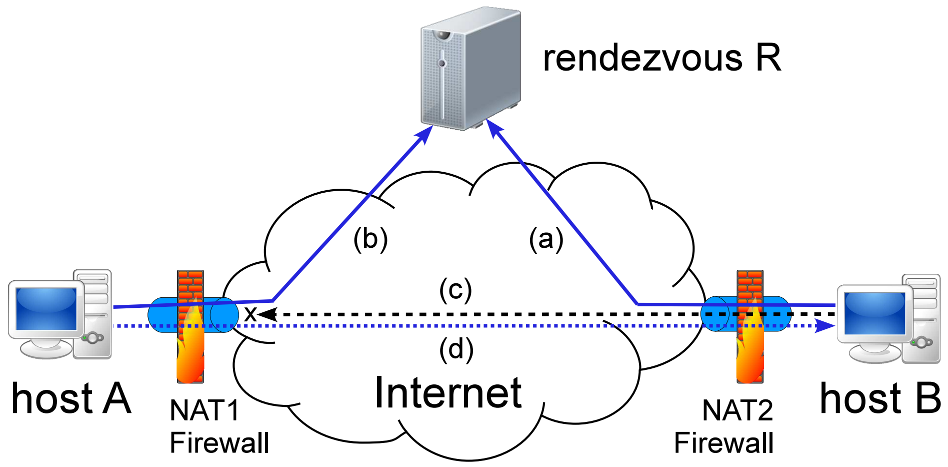
Source: Elaborated by authors.

it is important to highlight that although TCP hole punching (ROSENBERG, 2010) is similar, it does not work in several scenarios due to signalling and closed loop issues of the TCP.

UDP hole punching (SRISURESH et al., 2008) is a technique used to establish a direct UDP session between two hosts both behind NAT/firewall. The approach relies on a rendezvous server that must be previously known by the two hosts and is not behind NAT/firewall. The rendezvous assists in the establishment of the session, but is not used later.

Figure 3 shows how UDP hole punching works. Consider two hosts **A** and **B**, both behind NAT/firewall. Both hosts must have an active session with rendezvous **R**. When a host sends a UDP packet to the rendezvous in order to start a session, the rendezvous stores the host's public EP. Then, as shown in Figure 3 the following steps are executed: (a) **host B** communicates with **rendezvous R**; (b) **host A** communicates with **rendezvous R**, and finally (c) a UDP packet is sent from **host B** to **host A** in order to open a hole on **NAT2**, which is followed by (d) a UDP packet sent from **host A** to **host B** in order to open a hole on **NAT1**.

Figure 3 – UDP hole punching



Source: Elaborated by authors.

Thus the steps executed to open a hole on the NAT devices are:

1. **A** sends a message to **R** informing the address of **B**.

2. **R** sends a response to **A** with the public EP of **B**, and sends a packet to **B** with the public EP of **A**.

3. Each host, upon receiving the packet from **R**, begins to periodically send UDP packets to the EP of the other, using the same local port used in the session with the rendezvous. Eventually the holes on both sides will be opened.

4. In parallel with step **3**, each host listens for UDP packets from the other. When a packet arrives from the other host, a response is sent, and the procedure is completed successfully.

After these steps, both hosts can communicate freely using their known public EP's as if they were not behind NAT. As a side effect, this process may also open holes on firewalls at both ends as shown next.

Stateful firewalls employ the same principle as NAT for allowing packet flows, i.e. as the firewall processes the first packet of a given flow originated inside the network it creates state information that allows the destination host to send back packets to the protected network. The consequence is that stateful firewalls suffer from the same security fragility as NAT: any internal host can independently open holes to provide services that can be accessed from the rest of the Internet.

The approach we describe in the next section provides a functionality that is similar to Teredo's (HUITEMA, 2006). However our approach works either with IPv4 or IPv6. Although the main purpose of Teredo is just to provide IPv6 connectivity it also traverses NAT and firewalls.

Most recent research work on NAT traversal are related to IoT (Internet of Things). In (NOVO, 2018) tackles the fact that devices may suffer from resource constraints and may not be able to implement current NAT traversal architectures. A NAT traversal strategy based on SDN (Software Defined Network) technology to be applied in an IoT setting is presented in (WANG et al., 2019). We highlight that the authors do not even mention the security issues involved.

3 A STRATEGY FOR NAT & FIREWALL TRAVERSAL

We now describe in detail a strategy to enable transparent and direct communication between application processes using any transport protocol running on hosts behind NAT/firewall. The strategy is based on the use of virtual interfaces for tunneling and on the automatic establishment of IPv6 tunnels with the help of UDP hole punching. Although the strategy we describe employs IPv6, it also works with IPv4. The IPv6 traffic is encapsulated within UDP datagrams, which are sent across the IPv4/IPv6 Internet.

The strategy relies on two types of processes: clients and rendezvous. A client is a process that runs at the end-hosts to assist the NAT/firewall traversal. A rendezvous is a process running at a host not behind NAT/firewall whose main function is to help two clients to establish a UDP hole punch session. The interaction between clients and rendezvous is detailed next.

It is assumed that every client keeps a list of rendezvous. Initially, the client sends a UDP datagram to login to a rendezvous, and waits for a response for a limited time interval. The datagram can be retransmitted a configurable number of times until a response is received, or the client gives up. A rendezvous can deny the login request and send back a list of other rendezvous that the client can try. In other words, a rendezvous can redirect clients to other rendezvous. After successfully logging in at a rendezvous, the client sends UDP datagrams periodically in order to keep the UDP session active.

When a rendezvous accepts the login of a client, it chooses an IPv6 address that will be used by that client. Upon receiving the address, the client creates an IPv6 virtual tunnel interface using the Universal TUN/TAP driver [5]. This interface is not tied to any physical network card, but instead forwards the traffic received from the operating system to a user-specified process. From the application's point of view, the virtual network interface works as a conventional IPv6 interface. Furthermore, the application process can send/receive packets through the virtual interface, as if they had come from a physical network card.

As mentioned above, a rendezvous is a process that is not behind NAT/firewall. A rendezvous is a UDP server that listens for UDP datagrams coming from a client. A rendezvous keeps a table with information about all the clients with active sessions. Each table entry has the following data: the client's EP public IPv4, the client's IPv6 address and an estimated timeout. The timeout field records the time the rendezvous received the last message from the client. Using the timeout information it is possible to remove inactive sessions from the table, i.e., those inactive for a time interval larger than a threshold. It is easy to extend this strategy to allow multiple rendezvous.

4 EXPERIMENTAL RESULTS

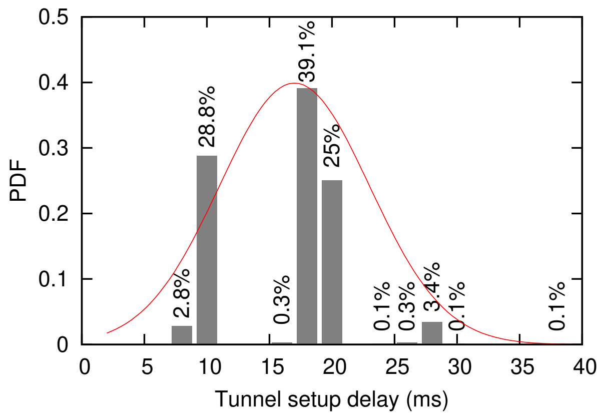
In this section we present two sets of experimental results. We first evaluate the overhead of using NAT traversal on the communication. Then we compare the strategy described with the alternative approach based on relay servers. All experiments were performed in a testbed that allowed us to control the routing complexity as well as to configure the delay of each link.

Initially, we measured the time required to setup the tunnel between the endpoints. Figure 4 shows that the mean delay to setup a tunnel is about 15 ms and none of the 1,000 tests executed required more than 40 ms.

Next, we evaluated the end-to-end delay *after* the tunnel had been created. Table 1 presents the RTT (Round-Trip Time) measurements obtained considering endpoints behind NATs and using direct routing. We have used the `netem` tool to create three different scenarios with different end-to-end delays: no extra delay, a fixed extra delay of 50 ms, and a

[5]http://vtun.sourceforge.net/tun/

Figure 4 – Time to establish tunnel between hosts behind NAT/Firewall



Source: Elaborated by authors.

variable extra delay with mean of 50 ms. RTT values were collected using `ping`/`ping6`, generating 1,000 probes (one per second) of 1400 bytes in each test. As can be seen in Table 1, the overhead is very low.

Table 1 – RTT between end systems

	Direct routing		**Behind NAT**	
Scenario	**Mean (ms)**	**Std. dev.**	**Mean (ms)**	**Std. dev.**
No delay	1.94046	1.14740	2.05956	0.50226
Fixed 50 ms	52.22857	0.48930	52.84094	0.66293
Variable 50 ms	53.99574	3.40016	54.05368	2.48792

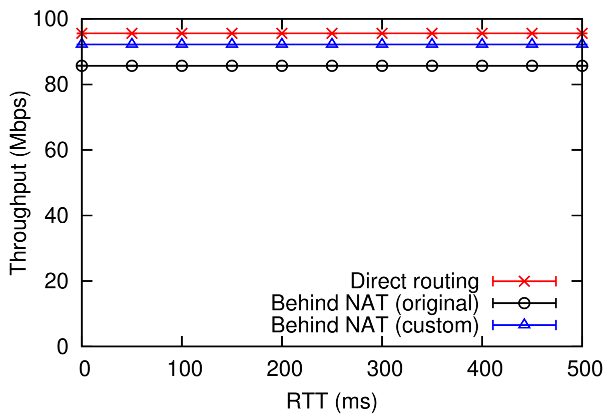
Source: Elaborated by authors.

A sequence of experiments were then executed in order to evaluate the performance of the strategy under a heavy throughput demand. Each experiment lasted for 60 s, and was executed 30 times for each configuration. The mean values are presented together with the confidence interval of 95%. This confidence interval is barely visible in the figures.

Figure 5 shows the throughput of UDP as a function of the end-to-end delay measured as the RTT. A higher RTT should not affect the UDP throughput, since the sender does not depend on the receiver feedback. However, this test is important to evaluate the overhead of the additional headers employed by the strategy and confirm the expected behavior. The `Iperf` tool was setup to generate 100 Mbps, i.e., the bottleneck throughput of the testbed. With direct routing a mean throughput of 95.6 Mbps was achieved, which is close to the maximum possible throughput, discounting UDP+IPv4 headers. Without any tuning, we reached a throughput value of approximately 85.7 Mbps, which was a bit lower than expected, and is caused by the extra headers employed, plus fragmentation. After adjusting the MTU of the virtual interface, the UDP throughput reached the unsurprising mean value of 92.2 Mbps.

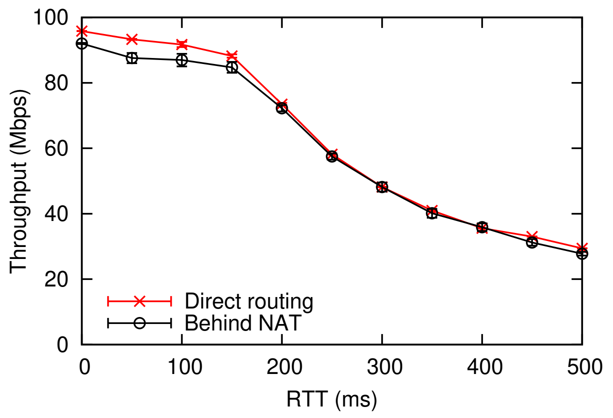
Figure 6 shows the throughput of TCP as a function of the RTT. Again, there is a little difference between direct routing and the NAT/firewall traversal technique when the RTT is below 200 ms, in this case the bandwidth-delay product (BDP) still has a minor impact on the TCP performance. When the RTT is greater than 200 ms, there is virtually no difference since BDP is dominant in the TCP performance results.

Figure 5 – UDP throughput as a function of the RTT



Source: Elaborated by authors.

Figure 6 – TCP throughput as a function of the RTT

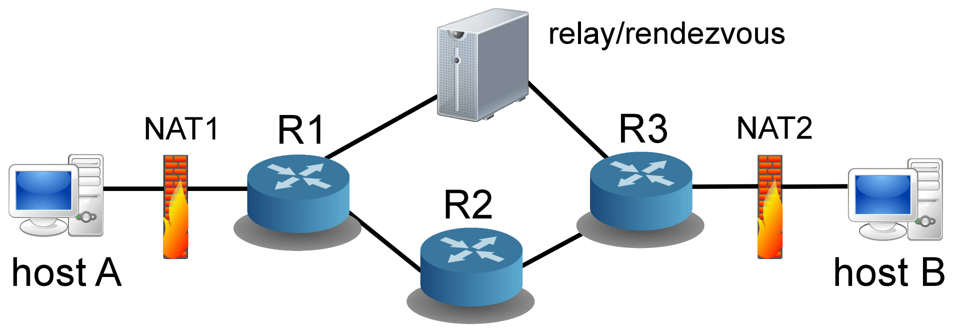


Source: Elaborated by authors.

The final result we present is a comparison with the relay server approach. The results show that the communication performance can be severely degraded when a third party is required for routing purposes. `Iperf` was again used to generate TCP traffic between end-hosts.

Figure 7 shows the environment used in the experiments. The path **NAT1-R1-R2-R3-NAT2** is the default route between **host A** and **host B**. The indirect path **NAT1-R1-RR-R3-NAT2** is used when the relay server is employed. We have built three scenarios: 1) the indirect path has characteristics that are similar to the default path, 2) the indirect path has a larger delay than the default path, and 3) the indirect path has less bandwidth than the default path. The first scenario is rarely found in the Internet. Results of these experiments are summarized in Table 2.

Figure 7 – Comparison with the alternative strategy using a relay server



Source: Elaborated by authors.

As can be seen in Table 2, the reversal strategy exhibits a stable performance in different scenarios. In the low delay scenario, the RTT of default route is 10 ms, while for the indirect path it is 100 ms. In the high capacity scenario, the bottleneck throughput of default route is 100 Mbps and indirect path is limited to 10 Mbps. Since only some control packets are exchanged between a rendezvous and a client, the rendezvous does not become a bottleneck to the system. A relay server remains an active part of the communication and thus has a higher impact on the performance.

Table 2 – Comparison between reversal technique and relay server

	Reversal Technique|		**Relay Server**	
Scenario	**Mean (Mbps)**	**Std. dev.**	**Mean (Mbps)**	**Std. dev.**
Similar env.	88.490	0.596	89.903	0.617
Low delay	88.490	0.596	86.587	3.098
High capacity	88.490	0.596	9.282	0.012

Source: Elaborated by authors.

Our results confirm that different types of applications can be deployed on top of NAT traversal techniques.

5 CONCLUSION

NAT traversal techniques are widely used. However, as a side effect, communicating with those techniques also involves opening firewall holes. In this way a security breach can be created as unauthorised services are made available from supposedly secured networks. The consequences can be severe to organizations and individuals connected to the Internet. In this article we described a simple approach to deploy NAT/firewall traversal and show experimentally that it is very efficient.

The challenges derived from this fact are far from trivial to solve. Although NAT and firewall have had a deep impact on the universal transparency of the Internet, they are key security technologies without which private networks would certainly become critically vulnerable. Future work includes the investigation of techniques that keep the functionality of NAT traversal, but are able to detect malicious/unauthorized usage.

REFERENCES

BODEI, C. et al. Language-Independent Synthesis of Firewall Policies. In: 2018 IEEE EUROPEAN SYMPOSIUM ON SECURITY AND PRIVACY (EUROS P). ee: ed, 2018. p. 92–106.

CARPENTER, B. **Internet Transparency**. 2000. <http://www.ietf.org/rfc/rfc2775.txt>.

FORD, B.; SRISURESH, P.; KEGEL, D. Peer-to-peer communication across network address translators. In: USENIX ANNUAL TECHNICAL CONFERENCE. **(ATEC)**. [S.l.], 2005.

FREED, N. **Behavior of and Requirements for Internet Firewalls**. 2000. <http://www.ietf.org/rfc/rfc2979.txt>.

GARRETT, T. et al. Monitoring Network Neutrality: A Survey on Traffic Differentiation Detection. **IEEE Communications Surveys & Tutorials**, v. 20, n. 3, p. 2486–2517, 2018.

HERRY, H. et al. Peer-to-peer secure updates for heterogeneous edge devices. In: NOMS 2018 - 2018 IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM. [S.l.], 2018. p. 1–5.

HO, C.Y. et al. To call or to be called behind NATs is sensitive in solving direct connection problem. **IEEE Communications Letters**, v. 15, n. 1, p. 94–96, 2011.

HSU, F.-H. et al. Handover: A mechanism to improve the reliability and availability of network services for clients behind a network address translator. **Computers & Electrical Engineering**, v. 67, p. 159–169, 2018.

HUITEMA, C. **Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)**. 2006. <http://www.ietf.org/rfc/rfc4380.txt>.

MAHY, R.; MATTHEWS, P.; ROSENBERG, J. **Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)**. 2010. <http://www.ietf.org/rfc/rfc5766.txt>.

MÜLLER, A.; WOHLFART, F.; CARLE, G. Analysis and Topology-based Traversal of Cascaded Large Scale NATs. In: PROCEEDINGS OF THE 2013 WORKSHOP ON HOT TOPICS IN MIDDLEBOXES AND NETWOEK FUNCTION VIRTUALIZATION. [S.l.], 2013. p. 43–48.

NOVO, O. Making Constrained Things Reachable: A Secure IP-Agnostic NAT Traversal Approach for IoT. **ACM Transactions on Internet Technology (TOIT)**, ACM New York, NY, USA, v. 19, n. 1, p. 1–21, 2018.

PANASONIC. **NAT Traversal Service**. 2013. <https://panasonic.net/cns/psn/products/hdvc/nat/nat_traversal/index.html>. Last access: Aug-2013.

PRO, CommuniGate. **NAT Traversal**. 2013. <http://www.communigate.com/cgatepro/NAT.html>. Last access: Aug-2013.

ROSENBERG, J. **Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols**. 2010. <http://www.ietf.org/rfc/rfc5245.txt>.

ROSENBERG, J. et al. **Session Traversal Utilities for NAT (STUN)**. 2008. <http://www.ietf.org/rfc/rfc5389.txt>.

SALTZER, J. H.; REED, D.P.; CLARK, D.D. End-to-end arguments in system design. **ACM Transactions on Computer Systems (TOCS)**, v. 2, n. 4, p. 277–288, 1984.

SRIRAMA, S. N.; LIYANAGE, M. TCP Hole Punching Approach to Address Devices in Mobile Networks. In: 2014 INTERNATIONAL CONFERENCE ON FUTURE INTERNET OF THINGS AND CLOUD. [S.l.], 2014. p. 90–97.

SRISURESH, P.; EGEVANG, K.B. **Traditional IP Network Address Translator (Traditional NAT)**. 2001. <http://www.ietf.org/rfc/rfc3022.txt>.

SRISURESH, P.; FORD, B.; KEGEL, D. **State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)**. 2008. <http://www.ietf.org/rfc/rfc5128.txt>.

TSENG, C.-C. et al. Can: A context-aware NAT traversal scheme. **Journal of Network and Computer Applications**, v. 36, n. 4, p. 1164–1173, 2013.

WANG, H.; CHEN, C.; LU, S. An SDN-based NAT Traversal Mechanism for End-to-end IoT Networking. In: 2019 20TH ASIA-PACIFIC NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (APNOMS). [S.l.], 2019. p. 1–4.

WEBRTC. **WebRTC**. 2013. <http://www.webrtc.org/>. Last access: Aug-2013.