This is a postprint version of the following published document:

Mandalari, A.M., et al. NATwatcher: profiling NATs in the wild. In: *IEEE Communications Magazine*, 55(3), March 2017, Pp. 178-185

DOI: <https://doi.org/10.1109/MCOM.2017.1600776CM>

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

NATwatcher: Profiling NATs in the Wild

Anna Maria Mandalari, Miguel Angel Diaz Bautista, Francisco Valera and Marcelo Bagnulo

University Carlos III of Madrid

Abstract—Network Address Translators (NATs) are a commonplace in the Internet nowadays. It is fair to say that most of the residential and mobile users are connected to the Internet through one or more NATs. As any other technology, NAT presents upsides and downsides. Probably the most acknowledged downside of the NAT technology is that it introduces additional difficulties for some applications such as peer-to-peer applications, gaming and others to function properly. This is partially due to the nature of the NAT technology but also due to the diversity of behaviors of the different NAT implementations deployed in the Internet. Understanding the properties of the currently deployed NAT base provides useful input for application and protocol developers regarding what to expect when deploying new application in the Internet. The goal of this paper is to identify common NAT profiles in order to provide an overview of the current behavior of NATs. We develop NATwatcher, a tool to test NAT boxes using a crowdsourcing-based measurement methodology. We perform a large measurement campaign using NATwatcher recruiting over 700 users, from 65 different countries and 280 ISPs. We present the results after testing and profiling NAT products from over 120 vendors.

INTRODUCTION

Network Address Translators (NATs) were introduced back in the early 90s as a mean to cope with the incipient address depletion crisis. Together with Classless Interdomain Routing (CIDR), they successfully extended the lifetime of IPv4 from imminent depletion until very recently, when the Internet Assigned Numbers Authority (IANA) pool of IPv4 addresses finally ran out [1]. NATs are now a commonplace in the Internet and they are included by default in the Internet Access offerings for both residential and mobile customers.

NATs successfully extended the lifetime of the IPv4 indeed, but at a high cost: they hardcoded the client-server paradigm in the architecture of the Internet. The basic operation of a NAT relies on the creation of a mapping state between a private address and port couple and a public address and port couple. This state is created when a client using a private address initiates a communication with a server in the public Internet. Deploying applications that have an alternative paradigm, such as peer-to-peer applications, gaming or Voice-over-IP to name a few, that require hosts in the public Internet to initiate communications towards hosts behind a NAT is challenging and requires the use of the so-called NAT traversal techniques. These techniques are usually cumbersome and increase the latency, the traffic and the energy consumed by the endpoint.

While the aforementioned problem of supporting alternative application paradigms is fundamental to the nature of the NAT operation, it is exacerbated by the myriad of behaviors of the different NAT implementations deployed in the Internet.

Different NATs use different criteria to create, preserve and remove their internal mapping state and have different filtering and forwarding rules. This severely complicates the job of applications willing to manage the NAT state in order to enable alternative communication models other than the client server one as they need to cope with all possible NAT flavours.

In order to achieve a more deterministic behavior from the NAT boxes, the Internet Engineering Task Force (IETF) produced a number of specifications defining the requirements that NATs should follow when creating, preserving and removing their internal state as well as some recommendations in terms of the different filtering and forwarding policies that NAT should implement. In particular, the IETF released NAT behavioral requirements for handling TCP traffic [2], UDP traffic [3] and ICMP packets [4]. The goal of these requirements is to achieve a more deterministic behavior of NATs and hence significantly simplifying the job of deploying new application paradigms in the Internet, fostering innovation and competition. However, since these IETF standards specify the internal behavior for a NAT, it is far from trivial to assess whether NATs are following the recommendations and to the best of our knowledge there is no information about the prevalence of NAT boxes that honor the IETF specifications.

Due to the difficulties that are inherent to perform large-scale measurements in real residential environments, to date, the few studies that are available have performed testing of different NAT devices in a lab environment. In [5] authors study the configurations of 34 different home gateway models, analyzing the processing of various TCP and IP options and measuring the success of some network protocols when traversing NATs (i.e., STUN [6], TURN [7] and ICE[8]). In [9] they perform a lab study of the support of a number of features such as mapping, filtering, hairpinning, on 42 NAT device models. While these studies provide some information about the capabilities of the different NAT boxes, they provide limited information about the actual behavior of the deployed NAT devices in the Internet. This is so because the behavior of the deployed NAT base also largely depends on the configuration of the NAT boxes and in the popularity of the different NAT products. The goal of this paper is to provide hard data about the behavior of the currently NAT base currently deployed in the Internet.

The contribution of this paper is three-fold. First, we design and develop NATwatcher, a tool to measure key aspects of the behavior of the deployed NAT base, along with a measurement methodology based on crowdsourcing that allows us to perform large scale measurement using NATwatcher.

Second, using the proposed methodology, we perform a large measurement campaign and we deploy NATwatcher in over 700 measurement points, building a large data set describing the behavior of over 700 NAT boxes from 65 different countries and 280 ISPs, testing over 120 different NAT vendors. Finally, we mine the obtained measurement results to identify common NAT profiles, providing valuable data for application developers about the ground truth of NAT behavior in the current Internet. We find that a large majority (80%) of the NAT boxes we tested follow the IETF recommendations for 11 out of the 17 of the considered features. We also observed that about half of the NAT devices we tested exhibit the exact same behavior for all the features we tested. While the other half of the devices use a variety of configurations.

METHODOLOGY AND SETUP

This section describes our methodology to classify NATs relying on crowdsourcing platforms to perform Internet-wide measurements.

Crowdsourcing Platform

Crowdsourcing platforms are online portals that allow employers to recruit workers from around the world to perform simple tasks, normally achievable in a few minutes, called *microjobs*. Each task contains a brief description of the work to be done and how the employers will verify the completion of the task. Once these tasks are defined, a campaign is run in the platform and every worker can apply to participate in the campaign and do the corresponding task. When the task has been completed the number of times required by the employer, the campaign is closed. Employers must then verify that the task has been completed by each worker and pay them accordingly. Crowdsourcing platforms traditionally are focused on the human element (i.e., to test psychological profile or to perform tests that machines could not do). We expand the usage of these platforms to run Internet wide measurements. In particular, we request workers around the world that are using a NAT to connect to the Internet to run out NATwatcher tool in order to characterize the behavior of their respective NAT.

NATwatcher: Methodology Overview

NATwatcher¹ is the tool we build to detect the characteristics of a NAT using a number of active tests. Figure 1 summarizes the operational setup of NATwatcher. In a nutshell, the worker downloads and executes the NATwatcher application. The NATwatcher application automatically executes the tests to characterize the NAT behavior, by sending different combinations of packets to our Measurement server deployed in the public Internet and processes the response packets sent from the server back to the NATwatcher client (step 1 in Figure 1). Once all the tests are completed the application sends the compiled measurement results to a collector server where they are stored (step 2 in Figure 1).

¹The source code of NATwatcher and the anonymized data sets are available at <http://www.it.uc3m.es/amandala/nat/natwatcher/>

Fig. 1: NATwatcher operational setup.

We designed NATwatcher to be suitable to be used in a crowdsourcing environment. In particular, we designed it to be as simple as possible in order to be tractable for a large number of workers. Workers are not required to have any technical background. Workers just need to download and execute the NATwatcher application and all data are automatically collected and reported to our collector server without further worker intervention.

In order to reach a high number of workers, we developed three versions of NATwatcher, one for Android, one for Windows and one for Linux. We therefore create two crowdsourcing campaigns, *Android-based* and *Windows/Linux-based*, which we detail next.

Android-based Campaign: This campaign requires the worker to install our Android application.² The worker then just have to launch the application and all the tests are then run sequentially. In order to ensure that workers' Internet access is provided through a fixed line (and not 3G, 4G, etc.), the application is instrumented to run the *microjob* using the WiFi interface.

Windows and Linux-based Campaign: This campaign requires workers to download the application³ and run it from a Windows or Linux machine. The application takes care of all the measurements to be performed as in the Android app. We asked workers to perform the test from their own personal computers, using a fixed line.

We also deploy a measurement server, connected to the public Internet. This server implements the server side of the tests described below including the UDP tests, the UDP STUN tests and the TCP STUN tests.

NATwatcher Tests

According to our experience, a *microjob* offered in a crowdsourcing platform is less likely to be performed by a large number of workers if it takes more than 5 minutes. In order to minimize of execution time of the *microjob*, we carefully selected 17 key NAT characteristics that NATwatcher will test for. We developed all the tests in C programming language to be able to reuse the code through the different platforms (i.e.,

²<http://www.it.uc3m.es/amandala/nat/natwatcher/natwatcher.apk>

³<http://www.it.uc3m.es/amandala/nat/nat.html>

Android, Windows and Linux). All the tests are implemented crafting UDP, ICMP and TCP packets using Raw Sockets or Standard Sockets whenever possible.

UDP tests

- 1) *Mapping behavior*: The test verifies if the NAT assigns the same mapping for communications between a specific internal IP address and port and any external IP address and port for UDP packets. NATs can be classified in 3 groups according to their mapping behavior:
 - **Endpoint-Independent mapping**: The NAT reuses the port mapping for subsequent packets sent from the same private IP address and port to any public IP address and port.
 - **Address-Dependent mapping**: The NAT reuses the port mapping for subsequent packets sent from the same private IP address and port to the same public IP address, regardless of the external port.
 - **Address and Port-Dependent mapping**: The NAT reuses the port mapping for subsequent packets sent from the same private IP address and port to the same public IP address and port while the mapping is still active.

The test is as follows: we first send two STUN *binding requests*⁴ to two different public addresses of our STUN server⁵. We compare the address and port returned in the two STUN responses received. If both the addresses and the ports match then we conclude that the mapping behavior of the NAT is *Endpoint-Independent*. If this is not the case, we send a third binding request to our STUN server using the primary addresses used before and a different port. If the address and port reported in the STUN response is the same than the one reported before when using the primary address and a different port, the NAT is *Address dependent*, else the NAT is *Address and Port-Dependent*.

- 2) *Filtering behavior*: The test detects the filtering behavior of the NAT. When an unsolicited packet is received from the Internet, the NAT applies filtering rules that can be classified as follows:
 - **Endpoint-Independent filtering**: The NAT forwards any packets destined to an internal host as long as a mapping exists.
 - **Address-Dependent filtering**: The NAT filters packets received from a specific external host to a specific internal host if the internal host has not previously sent any packet to that specific external host's IP address.
 - **Address and Port-Dependent filtering**: The NAT filters packets received from a specific external host to a specific internal host if the internal host has not

⁴The STUN protocol operates as follows: the client located behind the NAT sends a binding request message to a STUN server. The STUN server replies with a response message containing the IP address and port of the client in its payload, as observed from the server.

⁵A STUN server has two public IP addresses, the primary one and the alternate one.

previously sent any packet to that specific external host's IP address and port.

The test is as follows: We send a binding request to the primary address of our STUN server with a *change port* and *change address* attributes. These binding request attributes solicit the server to send the response from the alternate IP address and port. If the client receives the response, then the filtering behavior of the NAT is *Endpoint-Independent*. If not, we send a third binding request to the primary address with only *change port*. If the client receives a response then the NAT is *Address-Dependent Filtering*, if not it is *Port-Dependent Filtering*.

- 3) *Port Preservation*: This test checks if the NAT leaves the port unchanged when creating a mapping. We send a packet using a particular local source port and then we compare the port number with the external bound port. If they match the NAT is performing port preservation. If they do not match, it is possible that the NAT does not implement port preservation, but it is also possible that the specific port used as source port was already in use in another mapping (from another internal host). Because of this limitation, this test only provides a lower bound to the number of NATs that implement port preservation. It would be possible to increase the accuracy of this test by repeating the test several times. However, this would increase the time budget of the test, which we cannot afford.
- 4) *Hairpinning support*: Hairpinning enables a host in the home network to access another host in the home network using the external IP address. In order to check it we send two packets: the first one to discover our mapped address using STUN protocol and the second one from a different source port and towards the discovered mapped address. If we receive the response it means the NAT supports hairpinning.
- 5) *Supporting the "do not fragment" flag*: the NAT should properly generate the "ICMP fragmentation needed" message when a packet is received with the "do not fragment" flag set and discard that packet. Our application sends a packet with the Do not Fragment flag set and waits for the reception of the correspondent ICMP message.
- 6) *Out-of-order UDP fragments*: When packets are fragmented and received out of order some NATs are required to try to reassemble and then forward the packet. In order to check this behavior, we send from our server disordered fragments and check their reception in the application.
- 7) *Mapping lifetime over 2 minutes*: NATs are required to maintain UDP mappings during no less than 2 minutes. In order to check this, we send a first binding request to our server and after 2 minutes we send another binding request using a different port but specifying that the response should be sent to the previous binding. If the

Outbound refresh behavior: According to the UDP requirements, outgoing packets must refresh the existing binding. This test is similar to the previous one, except that an additional outgoing packet is sent one minute after the initial binding request and the second binding request is sent after 3 minutes.

- 1) *Reply/Request of ICMP packets*: The test verifies if the NAT supports simple ICMP Reply/Request message exchange by sending a query and waiting for the answer.
- 2) *Supporting of ICMP Destination Unreachable*: This test verifies the NAT is capable of forwarding ICMP Destination Unreachable messages generated as a response to a previous UDP packet. We send an UDP packet to our server and the server replies with Destination Unreachable behavior. We then verify if the ICMP error is received by the NATwatcher client.
- 3) *Supporting of ICMP Time Exceeded*: Same as the previous one but for ICMP Time Exceeded messages.
- 4) *Supporting error packet hairpinning*: NATs are required to support hairpinning for error messages. NATwatcher checks this by sending two packets, the first one to discover our mapped address using STUN protocol and the second one is an ICMP Echo Error message sent to the mapped address from a different port. The NATwatcher verifies the reception of the error message.

For the first four tests enumerated below, we use the same methodology than for UDP tests, but using TCP packets.

- We implement all the tests described before in the Linux and Windows versions of NATwatcher. However, 9 of the 17 tests need root permissions for their execution and unfortunately, Android devices are not commonly rooted. For this reason we implement only 8 over 17 NATwatcher tests in the case of the Android app. Specifically we implement test number 1, 2, 4, 7 and 8 for UDP NAT behavior and tests number 1, 2 and 4 for TCP behavior.

In this section, we present the collected data set, and we analyze the obtained results.

We defined two campaigns, one for the Windows/Linux-based application (each worker attempts 17 tests) and the other one for Android (each worker attempts 8 tests) and both were available for 28 days during June 2016. Overall, we recruited 781 workers: 170 workers participated in the Windows/Linux-based campaign, while 611 workers participated in the Android-based one.

Overall, workers are distributed among 65 countries (cf. Figure 2). The devices cover more than 280 ISPs for a total of 120 different NAT vendors.⁶

All the workers that we recruited through the crowdsourcing campaigns, are properly informed about the details so that they are able in time to make a choice of whether or not to participate in the test. This information has been carefully written in a way that was understandable to the people who have been approached as participants (non-technical workers). The source code we used to perform the tests is also provided to the workers so that they can check it.

⁶We use the WHOIS database to retrieve the ISP from the public IP address of each NAT and we use the MAC address of the NAT to retrieve the vendor.

TABLE I: IETF NAT requirements compliance results, based on the NATwatcher test.

Protocol	NATwatcher tests	NAT behavior	Number of devices(%)
UDP	Mapping behavior	Endpoint-Independent mapping	80.9%
		Address-Dependent mapping	0.7%
		Address and Port-Dependent mapping	18.4%
	Filtering behavior	Endpoint-Independent filtering	14.63%
		Address-Dependent filtering	1.79%
		Address and Port-Dependent filtering	83.56%
	Port Preservation	Yes	78.2%
		No	21.8%
	Supporting hairpinning	Yes	15.5%
		No	84.5%
	Supporting the “not fragment” flag	Yes	100%
		No	0%
ICMP	Supporting receiving UDP fragments out-of-order	Yes	100%
		No	0%
	Mapping lifetime over 2 minute	Yes	12.5%
		No	87.5%
	Outbound refresh behavior	Yes	86.03%
		No	13.97%
	Reply/Request of ICMP packets	Yes	84.8%
		No	15.2%
TCP	Supporting of ICMP Destination Unreachable	Yes	69.6%
		No	30.4%
		Yes	91.22%
	Supporting of ICMP Time Exceeded	No	8.78%
		Yes	83.04%
	Supporting error packet hairpinning	No	16.96%
TCP	Mapping behavior	Endpoint-Independent mapping	78.4%
		Address-Dependent mapping	0.6%
		Address and Port-Dependent mapping	21%
	Filtering behavior	Endpoint-Independent filtering	12.5%
		Address-Dependent filtering	4.85%
		Address and Port-Dependent filtering	82.65%
	Port Preservation	Yes	78.2%
		No	21.8%
	Supporting hairpinning	Yes	15.5%
		No	84.5%
	Mapping and ICMP packets	Mapping is maintaining after receiving	29.8%
		Mapping is not maintaining after receiving	70.2%

General Results

Table I summarizes the results for both campaigns. For each test we show the percentage of NATs that follow a particular behavior. We color in grey the IETF recommended behaviors.

We can see that for 11 out of 17 there is wide compliance with the IETF sanctioned behavior (for these 11 tests, more than 80% of the analyzed NATs follow the recommended behavior). For the remaining 6 tests, the large majority of tests NAT boxes do not follow the IETF recommendation.

Most of the measured NATs models implement UDP and TCP Endpoint-Independent mapping, support UDP not fragment flag and receiving UDP fragments out-of-order, moreover most of them follow all the ICMP requirements. All these requirements are reported as a MUST in [2], [3] and [4]. For the rest of the tests the success rate decreases dramatically. In particular it is interesting to see less than 16% of the devices fulfill the requirements on UDP and TCP Endpoint-Independent filtering, hairpinning support and mapping lifetime over 2 minutes. The Endpoint-Independent filtering is not a mandatory requirement, but it is highly recommended if transparency is a priority. If security is the priority, NAT should follow an Address-Dependent filtering, but as results show, the majority of NATs set an Address and Port-Dependent filtering. Applications such as online gaming for instance would not benefit from this kind of behavior.

Hairpinning and binding lifetime over 120 seconds are two

mandatory requirements for UDP. Most of the NATs do not support these features as the failure rate is higher than 84% for these tests.

Port preservation support is recommended particularly for outgoing TCP connections, in order to allow NAT port prediction. In our results, we find a lower bound of 78% of devices that use port preservation.

Results show that about 70% of NATs delete the TCP mapping if an ICMP error message is received affecting that specific mapping. This is strongly recommended against in [2] as it exposes the NAT to attacks relying on ICMP error messages to delete existing NAT bindings.

NAT Classification

In this subsection we identify the most common NAT configurations. For this analysis we only consider the 8 of the tests have been performed for all the tested NAT devices (i.e. the 8 tests that are executed in all the campaigns). We characterize a configuration through the different combinations of the results of these 8 tests. We identified 19 different configuration across the 781 NATs we tested. Only 11 out of these 19 configurations appear in more than 5 devices. We detail them in Table II. Overall 52.5% of devices implement the configuration number 1. The devices come from over 50 countries and 173 unique ISPs, for a total of 72 different vendors. The most common configuration includes Endpoint-

a) Devices ID in order of vendors

b) Devices ID in order of ISPs

Fig. 3: Configuration number vs. device ID.

Configuration Number	UDP mapping	UDP filtering	UDP hairpinning	TCP mapping	TCP filtering	UDP mapping over 2 min	UDP outbound mapping refresh	TCP hairpinning	Number of devices (%)
1	E.I.	A.P.D.		E.I.	A.P.D.	✓			52.5
2	E.I.	E.I.	✓	A.P.D.	A.P.D.	✓	✓		5.5
3	A.P.D.	A.P.D.		E.I.	A.P.D.				5.2
4	E.I.	E.I.		A.P.D.	A.P.D.				2.5
5	E.I.	E.I.	✓	E.I.	E.I.	✓	✓		2
6	E.I.	A.P.D.		A.P.D.	A.P.D.				1.8
7	A.P.D.	A.P.D.		A.P.D.	A.P.D.	✓			1.5
8	E.I.	E.I.		E.I.	E.I.				1.15
9	E.I.	A.D.		E.I.	A.D.				0.8
10	A.P.D.	A.P.D.		E.I.	A.D.	✓			0.8
11	E.I.	A.P.D.		E.I.	A.P.D.	✓			0.65

TABLE II: Most common NAT configurations.

Independent mapping for UDP and TCP and the filtering as strict as possible (Address and port dependent).

This is a clear trend towards security for the rest of configurations, as only 4 over 11 NAT configurations implement Endpoint-Independent filtering behavior.

Table II also shows that for both UDP mapping lifetime and the hairpinning, the majority of analyzed devices fail to comply with the IETF recommendations.

Vendors and ISPs

In this section we try to figure out if the adoption of a certain common NAT configuration depends on the vendor or on the ISP.

Figure 3 shows the configuration number for each tested device. The devices (in the horizontal axis) are ordered first by vendor (Figure 3 (a)) and second by ISP (Figure 3 (b)). Each vendor/ISP is delimited by a vertical line. Devices from vendors/ISPs present in our data set with not more than 2 devices are grouped after the last line on the right. We ordered the configurations by IETF requirement compliance (i.e., the configuration with higher RFC requirements compliance first and the one with less RFC requirements compliance

last). For example, configuration number 1 in Figure 3 is the configuration recommended in the different RFCs, which has UDP Endpoint-Independent mapping and filtering, TCP Endpoint-Independent mapping, supports hairpinning and the UDP mapping lifetime is over 2 minutes. We consider this configuration “open”, meaning that transparency is the top priority compared to security.

Apart from a clear trend to use configuration number 9, as mentioned earlier, we observe some “open” configurations for few vantage points in some vendors. This can be explained considering that users can change the basic configuration that vendors impose by default to accommodate the NAT to their own needs (i.e., activating hairpinning or Endpoint-Independent filter).

In Figure 3 (b), we identify 2 ISPs (identified by arrows) where the behavior for all NATs connected to these ISPs is very uniform, exhibiting one or two configurations. These two ISPs are mobile ISPs that provide 3G and LTE Internet access services. Given that NATwatcher only runs using the WiFi interface and not the cellular one, this means that the NATwatcher client was executed in a mobile phone connected to a hotspot which in turn was connected to the Internet using 3G or LTE. We believe it is safe to conclude that in these cases, the NATwatcher client was behind two cascaded NATs, the one from the Wifi access point and the one from the mobile operator. This means that when the client executes NATwatcher, the results reflect the superposition of both NATs along the path, reflecting the more restrictive behavior of the two, which is likely to be the one of the NAT of the mobile operator explaining the uniform behavior observed.

CONCLUSION

In this paper we presented NATwatcher, a tool that allows to shed some light on the NAT behavior in the wild. Using a crowdsourcing approach we provide insight into how vendors, ISPs or users configure and use NATs with respect to TCP, UDP and ICMP packets, providing useful data for designing

future applications. We presented the first large-scale measurement campaign of home NATs, based on data from 781 homes, from 65 countries and from 280 ISPs.

Our study demonstrated that about 80% of the tested NATs follow the IETF sanctioned behavior with respect to 11 tests over 17 (64% of tests). For the remaining 6 tests our findings show that only 13% of the NAT boxes follow the IETF requirements of filtering, hairpinning and mapping lifetime over 2 minutes. Moreover, we listed the 11 most common configurations, finding that the 52.5% of the tested NAT boxes use Endpoint-Independent mapping for UDP and TCP and Address and port dependent filtering, they do not support hairpinning or UDP mapping over 2 minutes. To the best of our knowledge, this is the largest dataset available describing the behavior of the deployed NAT base and we believe it would provide useful input for application and protocol designers aiming to make their applications and protocols to work across NATs.

REFERENCES

- [1] “Potaroo Network Tool.” <http://www.potaroo.net/tools/ipv4/>, accessed: 2016-07-14.
- [2] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, “NAT Behavioral Requirements for TCP,” Tech. Rep., 2008.
- [3] F. Audet and C. Jennings, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP,” Tech. Rep., 2007.
- [4] P. Srisuresh, B. Ford, S. Sivakumar, and S. Guha, “NAT Behavioral Requirements for ICMP,” Tech. Rep., 2009.
- [5] S. Hättönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, “An Experimental Study of Home Gateway Characteristics,” in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 260–266.
- [6] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, “Session Traversal Utilities for NAT (STUN),” Internet Requests for Comments, RFC 5389, October 2008.
- [7] J. Rosenberg, R. Mahy, and C. Huitema, “Traversal Using Relay NAT (TURN),” October 2003. Internet-Draft (Work in Progress), Tech. Rep., 2005.
- [8] J. Rosenberg, “Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/answer Protocols,” Tech. Rep., 2010.
- [9] C. Jennings, “Nat classification test results,” *Work in Progress*, 2007.