NAT-Compatibility Testbed: An Environment to Automatically Verify Direct Connection Rate

Cheng-Yuan Ho, *Member, IEEE,* Fu-Yu Wang, Chien-Chao Tseng, *Member, IEEE,* and Ying-Dar Lin, *Senior Member, IEEE*

***Abstract*—In this article, an NAT-compatibility testbed is designed to automatically conduct the repeated experiments, collect the test results, and verify the direct connection rate (DCR) of any NAT traversal programs. Achieving a high DCR is important because using relays may unnecessarily increase the bandwidth cost, processing load of the relay servers, and the end-to-end packet delay. The NAT-compatibility testbed is constructed with 4 components: two peers, an automatic execution mechanism, NAT devices, and signaling/intermediate servers such as STUN, TURN, and SIP. It is also called the fully meshed testbed because the test result of all NAT combinations is a square. It measures the DCR of ICE, KeyStone, and PJNATH as 53.7%, 59.87%, and 50.93%, respectively. Experimental results show that asymmetric and unexpected direct connectivity check results occur in the real Internet. In order to enhance NAT traversal capability, the findings, like "port prediction" and "call-role sensitivity problem," are also described in the experimental results.**

***Index Terms*—NAT, NAT traversal, direct connection, testbed.**

I. Introduction

NETWORK address translation (NAT) is a common solution to the IP address depletion problem of Internet Protocol version 4 (IPv4). The basic NAT concept is that it maps an address in one realm to an address in another, while providing transparent routing for the hosts behind the NAT called *internal hosts* [1]. Traditional NATs adopt network address port translation (NAPT) and can translate many internal network addresses and their transport ports into a single external network address and many transport ports, called *mapped-addresses*. Thus, with NAPT, an NAT can serve many internal hosts in a private network with a public IP address.

However, an NAT also introduces the NAT traversal problem. A host *external* to an NAT can not originate connections to an internal host unless the NAT has previously established a mapped-address for the internal host. Moreover, an NAT may employ filtering rules to block unauthorized inbound traffics. Hence, it is why NAT traversal is an important problem today.

Several protocols have been proposed to facilitate NAT traversal. Session Traversal Utilities for NAT (STUN) [2], [3] provides a protocol that allows an internal host to *discover* the presence and type of its NAT and mapped-address. Traversal Using Relays around NAT (TURN) [4] is a protocol that allows two hosts to control the operation of the relay and exchange packets with each other through the relay. Interactive Connectivity Establishment (ICE) [5] makes use of both STUN and TURN for any two peers to *discover* and *exchange* three *candidate* transport addresses including *host* (itself), *mapped* (on the NAT), and *relayed* (on the relay server) transport addresses. Furthermore, it also provides a connectivity check algorithm that two hosts can use to determine and agree on which *candidate pair* to use for NAT traversal. Therefore, ICE is currently the most promising NAT traversal technique.

The concept of ICE is used in many NAT traversal programs, such as PJSIP NAT Helper (PJNATH) [6] and KeyStone [7]. PJNATH is an open source library while KeyStone is a commercial program of Teltel. However, experiment results, in the NAT-compatibility testbed, show that direct connection rates (DCRs) of NAT traversal programs are different. Moreover, the NAT-compatibility testbed is constructed with 4 components: two peers, an automatic execution mechanism, NAT devices, and signaling/intermediate servers. Different NAT traversal ability may affect the test results; for example, ICE might fail to discover a direct connection that in fact exists between the two peers while KeyStone succeeded. Failure in direct connection tests suggests the use of a relay to forward packets on behalf of peers; however, using relays unnecessarily increases the bandwidth cost and processing load of the relay servers, and, even worse, the end-to-end packet delay, which is especially harmful to real-time P2P applications [8].

In this work, we present the design of our NAT-compatibility testbed and experiment results of ICE, KeyStone, and PJNATH, and summarize the findings. The remainder of this article is organized as follows. In Section II, we introduce NAT types and describe the design, attribution, and operations of our fully meshed testbed. Section III presents experiment results and summarizes the findings. Finally, we provide suggestions for further research in Section IV.

II. NAT Types and NAT-Compatibility Testbed

A. NAT Types

NATs can be classified into four types: *full cone*, *address restricted cone*, *port restricted cone*, and *symmetric* [2], according to the mapping and filtering rules[1]. All cone NATs have the mapping rule that *all* requests from the same internal transport address are mapped to the *same* mapped-address; whereas, in a symmetric NAT *each* request from the same internal transport address to a specific remote transport address is mapped to a *unique* mapped-address. A full cone NAT allows any external host to send packets to an internal host if the NAT already has a mapped-address for that internal host.

[1]In [3], Rosenberg et al. suggest that STUN remove four NAT types, but it will be difficult to explain our experimental results and findings. Therefore, we stay with the original classification in [2] for the clearer explanation.

Manuscript received September 12, 2010. The associate editor coordinating the review f this letter and approving it for publication was F.-N. Pavlidou.

C.-Y. Ho is with the D-Link NCTU Joint Research Center, National Chiao Tung University, Taiwan (e-mail: cyho@csie.nctu.edu.tw).

F.-Y. Wang is with the Network Benchmarking Lab (NBL), NCTU, Taiwan (e-mail: sagual@nbl.org.tw).

C.-C. Tseng (corresponding author) and Y.-D. Lin are with the Department of Computer Science, NCTU, Taiwan (e-mail: {cctseng, ydlin}@cs.nctu.edu.tw).

This work was supported in part by D-Link Co., Taiwan.

Digital Object Identifier 10.1109/LCOMM.2010.102810.101700

1089-7798/11$25.00 © 2011 IEEE

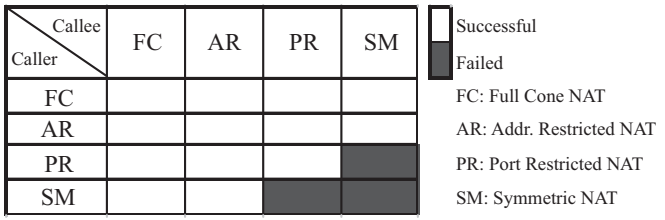

Fig. 1. Direct connectivity of ICE.

However, an address restricted cone NAT allows an external host with a particular IP address to send packets to a mapped-address only if the NAT had *previously* sent a packet with the mapped-address to the IP address. A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. A symmetric NAT adopts the same filtering rule as a port restricted cone NAT.

Accordingly, the ability of two hosts behind different NATs to establish direct connection depends on which host *initiates* the communication. For example, suppose *host1* and *host2* are behind a full cone NAT and a port restricted NAT, respectively, and know each other's mapped-address by some signaling protocol such as Session Initiation Protocol (SIP) or STUN. *Host1* cannot establish a direct connection to *host2* if *host1* initiates the communication because the packets sent by *host1* will be dropped by the port restricted NAT of *host2*. However, because of the *loose* restriction rule of the full cone NAT, *host2* can establish a direct connection to *host1* by initiating the communication. ICE resolves the problem by having *each* of two hosts initiate a connectivity check for *each* candidate pair. Figure 1 summarizes the cases where ICE should establish a direct connection under various NAT combinations.

Among the various NAT types, port restricted and symmetric NATs employ the most stringent filtering rule. However, when both peers are behind the port restricted NATs they still can establish a direct connection because both can initiate a connectivity check for the same mapped-address pair, and *at least* one of them will pass through the NAT at the other end. On the other hand, when one peer is behind a symmetric NAT and the other peer is behind a port restricted or symmetric NAT, it is not easy for the two peers to establish a direct connection [9]. This is because, for each *outgoing* connection of an internal host, symmetric NATs assign a unique mapped-address.

B. NAT-Compatibility Testbed

Conducting the repeated experiments, collecting test results, and verifying the DCR of any NAT traversal programs can be done by hand, but this may waste a lot of manpower and time. For example, one needs to pull the power and wired lines out an NAT device and then plug them into the other NAT device for one NAT combination test. Therefore, an NAT-compatibility testbed is designed for automatically accomplishing above mentioned works. The testbed is constructed with 4 components: two peers, an automatic execution mechanism, NAT devices, and STUN, TURN, and SIP servers. Figure 2 shows the testbed. The functions of 4 components are as follows.

1) One peer plays as a caller and the other is a callee. Both two peers execute the same NAT traversal program to do the connectivity check to see whether they can find a direct connection in some NAT combination.

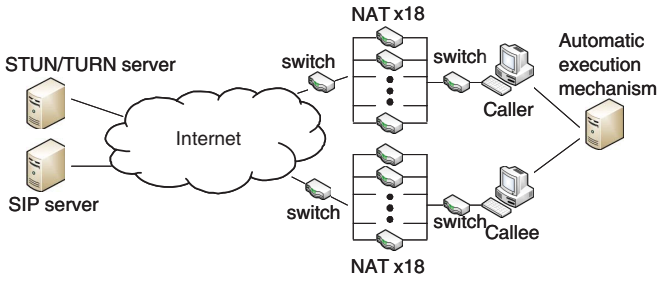

Fig. 2. NAT-compatibility testbed environment.

TABLE I
NAT BRANDS AND MODELS IN EXPERIMENTS

No	Brand	Model	No	Brand	Model
1	Dlink	Di-604	10	Edimax	Br-6204wg
2	Smc	Smcwbr14-g2	11	Linux	Iptables
3	Corega	Cg-barmx2	12	Linksys	Befsr41
4	Planex	Blw-54mr	13	Linksys	WRT150N
5	Smc	SmcWGBR-14n	14	Abocom	Fsm410
6	3com	3crwer 100-75	15	Asus	Rx3041
7	Belkin	F5d8231tw4	16	Netgear	Pr614
8	Draytek	vigor2104p	17	Zyxel	P334
9	Lemel	Lm-wlg6400	18	Freebsd	Pf

2) The automatic execution mechanism is to setup the private IP addresses, subnet masks, and gateways of two peers, assign which peer to be the caller or callee, and decide when the connectivity check begins automatically.

3) Different NAT devices lead to various NAT combinations. In this work, 18 NAT devices are used, and therefore, there are 324 NAT combinations in experiments. There are two reasons for wiring NAT devices like Fig. 2. One reduces the human careless plug-in errors which will affect experiment results, and the other is easy to extend or change the number and brand of NAT devices.

If a caller's request is to go through NAT 3, for example, the caller's gateway is set to the internal IP address of NAT 3. If it is to go through NAT 5 at the callee's side, its destination address is set to the callee's mapped-address which equals the external IP address and some transport port of NAT 5. The brand and model of NAT devices are shown in Table I.

4) STUN, TURN, and SIP servers are located in a public IP domain to provide services such as NAT mapped-address notification, registration, and relay, when requested by NAT traversal programs. All NAT traversal programs need to get NAT mapped-address from the STUN server, for instance, but would not use the TURN server to relay packets if a direct connection can be found.

Only the hardware environment of NAT-compatibility testbed cannot make experiments work, so the following is about how to enable the experiments and related steps. During each test, all procedures are controlled by the scripts in the automatic execution mechanism and packet flows are recorded for the future analysis. The steps for each test are as follows.

Step 1: Set up IP addresses, subnet masks, and gateways of two peers behind NATs sequentially (i.e., from NAT 1 to 18).

Step 2: Two peers execute the same NAT traversal program. Moreover, the IP addresses of the STUN/TURN and the SIP servers are filled, if necessary.

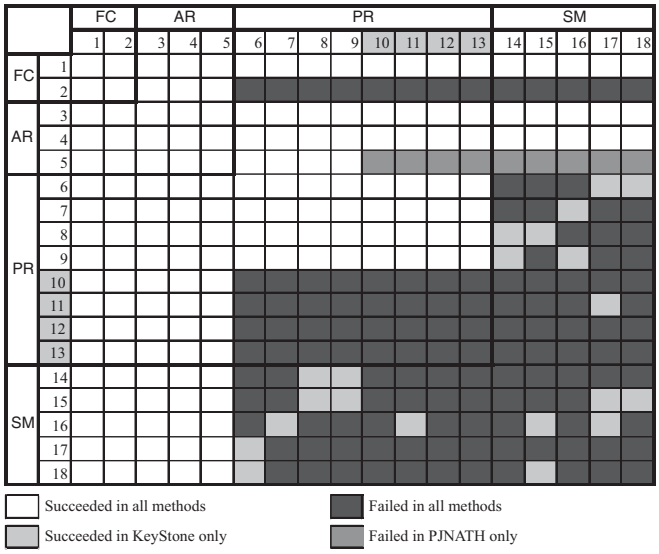

Fig. 3. Experimental results of ICE, KEYSTONE, and PJNATH.

Step 3: The caller makes a connection to the callee. The caller initiates a connection, and the callee accepts this. This connection is kept alive for 60 seconds.

Step 4: After 60 seconds, both peers not only end the connection but also shut down the NAT traversal program.

Step 5: Two peers stop recording packet flows and save to a separate packet trace file.

If two peers receive each other's packets with the *source* IP address and port number being the IP address and port number of each other's NAT device, then this connection is *direct*. Otherwise, this means that packets are relayed by the TURN server since the relay truncates the connection into two and changes the source address and port.

III. Experimental Results

Figure 3 shows experimental results of three NAT traversal programs: ICE, KeyStone, and PJNATH, respectively. From Fig. 3, we observe all of three NAT traversal programs have *asymmetric* direct connectivity check results. For example, when the callee is behind one of NATs 6 to 18 and the caller is behind NAT 2, they cannot establish a direct connection. However, they can communicate with each other directly while the caller is behind one of NATs 6 to 18 and the callee is behind NAT 2. This is because though NAT 2 is a *Linux-based* full cone NAT, it acts like a symmetric NAT for connections originated by an internal host if that host attempts to create several connections simultaneously within a short period.

Furthermore, among 18 NATs as shown in Table I, NATs 1 and 2 are full cone, NATs 3 to 5 are address restricted, NATs 6 to 13 are port restricted, and NATs 14 to 18 are symmetric. According to the direct connectivity of ICE (as shown in Fig. 1), two peers should establish a direct connection when the caller is behind one of NATs 10 to 13 and the callee is behind one of NATs 6 to 13. However, none of these three NAT traversal programs can let two peers communicate with each other directly. After analyzing pcap files, we observe that the caller's NAT not only drops the connectivity check request sent by the callee, but also assigns a *new* mapped-address for the request sent by the caller. This request originated from the new mapped-address will also be dropped by the port restricted cone or symmetric NAT at the callee's side because the NAT did not send packets to that IP address before. Such a problem is caused by the *connection tracking*, one feature built within the *Linux kernel*, and called *call-role sensitivity problem* [10]. Also, this kind of NATs is named Linux-based NATs in [10].

The behavior of PJNATH is similar to that of call-role sensitivity problem when the caller is behind NAT 5 and the callee is behind one of NAT 10 to 18, as shown in Fig. 3. However, compared to ICE and KeyStone, it seems programming errors or logical fallacies happen in the PJNATH because others can get direct connectivity check results.

Among three NAT traversal programs, only KeyStone has chance to make two peers establish a direct connection, although it may be asymmetric, when both are behind symmetric NATs. After tracing pcaps, we find KeyStone implements so-called *port prediction* mechanism [9].

From experimental results, we could observe that, out of the 324 possible combinations, the DCR of ICE, KeyStone, and PJNATH is 53.7% (i.e., 174 direct connections), 59.87% (194), and 50.93% (165), respectively. Furthermore, the DCR will be increased when the asymmetric direct connection problem and call-role sensitivity problem are solved or more powerful port prediction mechanism is developed.

IV. Conclusions

In this article, we designed an NAT-compatibility testbed which can automatically conduct the repeated experiments, collect test results and verify any NAT traversal programs' direct connection rate. Experimental results show that the fully meshed testbed really works and testifies the DCR of ICE, KeyStone, and PJNATH. From experimental results, the call-role sensitivity and asymmetric direct connection problems, and the port prediction mechanism are found. Furthermore, this testbed is not only to save manpower and time to automatically accomplish repeated works on UDP but also suitable for TCP because STUN, TURN, and ICE support TCP now.

References

[1] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)," IETF RFC 1631, May 1994.
[2] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) through network address translators (NATs)," IETF RFC 3489, Mar. 2003.
[3] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," IETF RFC 5389, Oct. 2008.
[4] J. Rosenberg, R. Mahy, and P. Matthews, "Traversal Using Relays around NAT (TURN): relay extensions to Session Traversal Utilities for NAT (STUN)," IETF RFC 5766, Apr. 2010.
[5] J. Rosenberg, "Interactive Connectivity Establishment (ICE): a protocol for network address translator (NAT) traversal for offer/answer protocols," IETF RFC 5245, Apr. 2010.
[6] PJSIP NAT Helper, URL: http://www.pjsip.org/
[7] KeyStone, URL: http://www.teltel.com/en/index.html
[8] H. Khlifi, J.-C. Gregoire, and J. Phillips, "VoIP and NAT/firewalls: issues, traversal techniques, and a real-world solution," *IEEE Commun. Mag.*, vol. 44, no. 7, pp. 93-99, July 2006.
[9] Y. Wang, Z. Lu, and J. Gu, "Research on symmetric NAT traversal in P2P applications," in *Proc. International Multi-Conference on Computing in the Global Information Technology (ICCGI)*, p. 59, 2006.
[10] C.-Y. Ho, C.-C. Tseng, F.-Y. Wang, J.-T. Wang, and Y.-D. Lin, "To call or to be called behind NATs is sensitive in solving direct connection problem," *IEEE Commun. Lett.*, to appear.