A New Method for Symmetric NAT Traversal in UDP and TCP

Yuan Wei
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo, JAPAN

Daisuke Yamada
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo, JAPAN

Suguru Yoshida
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo, JAPAN

Shigeki Goto
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo, JAPAN

{wei,daiski,yoshida,goto}@goto.info.waseda.ac.jp

ABSTRACT

This paper proposes a new method for Network Address Translator (NAT) Traversal in UDP. Several techniques have been proposed for traversing NAT or firewall boxes in UDP. These techniques can establish UDP communication between hosts behind NATs. However, existing NAT traversal methods, including Universal Plug and Play (UPnP), Simple traversal of UDP over NATs (STUN) and Teredo, cannot traverse symmetric NAT boxes. Our method uses a new port prediction method. It controls ports to traverse symmetric NAT boxes as well as other kinds of NATs. In addition, our new method can be extended for simple NAT traversal in TCP. The method is based on a new UDP hole punching technique.

We have tested nine working NAT products in our laboratory. The results show that our method can be practically implemented for successful NAT traversal for real use.

Keywords

NAT traversal, Symmetric NAT, UDP, P2P, Stateful Packet Inspection, TCP

1. INTRODUCTION

A network address translator (NAT) is a well-known, versatile tool that enables the reuse of IP addresses in the Internet. Using a NAT, we can convert private IP addresses to global IP addresses. However, a fatal problem can occur if an applications protocol includes an IP address as part of the *payload* of IP packets. This is because NAT translates IP addresses in the header properly, it cannot convert IP addresses in the payload. Examples of applications that suffer from this problem include Voice Over IP and Multimedia Over IP applications such as SIP [1] and H.323 [2] as well as online games.

Asia Pacific Advanced Network 2008,
4-8 August 2008, New Zealand.
Network Research Workshop 2008,
4 August 2008, New Zealand.

There have been many proposals to solve this problem. Several real-time multimedia applications, online games, and other applications that work properly across NATs have been developed using standard techniques such as Universal Plug and Play (UPnP) which has been adopted by many vendors [3]. Another example of a commonly used protocol is Simple Traversal of UDP (STUN) [4], which is an implementation of the UNilateral Self-Address Fixing (UNSAF) protocol [5]. Teredo realizes an UNSAF mechanism by tunneling IPv6 over UDP/IPv4 [6]. However, these proposals do not solve the problem completely because none of them can work successfully with all types of NATs.

This paper proposes a new method for NAT traversal, which is applicable to symmetric NATs as well as other types of NATs. Symmetric NATs are used when high security communication is required. For example, the most expensive router sold by a Japanese manufacturer, who sells nine types of routers in the market, is the one equipped with symmetric NAT functionality. Symmetric NATs are installed as routers in business enterprises and also as high-end routers for home use. Our new method is based on port *prediction*. It manipulates *port numbers* in order to traverse symmetric NATs successfully. We have conducted several experiments to evaluate the performance of our new method. The results show that our method can be practically implemented for successful NAT traversal. In addition, the new method can be also extended to develop a new method for NAT traversal in TCP.

Section 2 describes the various types of NATs. Section 3 surveys the existing methods of NAT traversal. Our new method is proposed in Section 4. Section 5 shows the results of our experiments and Section 6 concludes this paper.

2. TAXONOMY OF NATS

The study on the STUN protocol [4], use terms such as *Full Cone*, *Restricted Cone*, *Port Restricted Cone* and *Symmetric* to describe the different types of NATs. These NATs are discussed with reference to UDP only. We will mention TCP NATs briefly in Section 4.4.

2.1 Full Cone NAT

A *full cone* NAT is also known as a one-to-one NAT. Once an internal IP address and port are mapped to some external

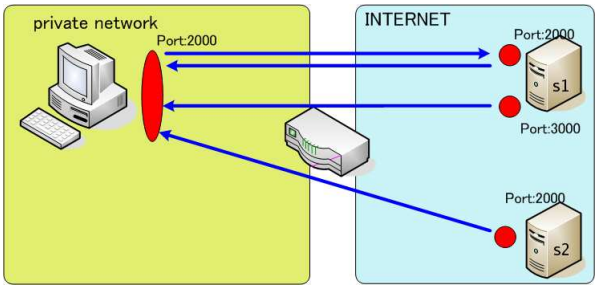

Figure 1: Full cone NAT

IP address and port respectively, all the packets with the internal IP address and port will be translated to the fixed external IP address and port. Furthermore, any external host can send a packet to the internal host by sending a packet to the mapped external address. The full cone NAT is illustrated in Figure 1.

2.2 Restricted Cone NAT

In the *restricted* cone NAT, all requests from an internal IP address and port are mapped to a fixed external IP address and port. It is similar to the full cone NAT except that unlike the full cone NAT, an external host $s2$ (with IP address x) can send a packet to an internal host only if the internal host has previously sent a packet to the IP address x through the restricted cone NAT. The restricted cone NAT is illustrated in Figure 2.

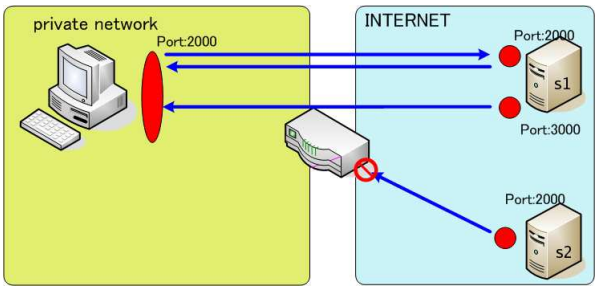

Figure 2: Restricted cone NAT

2.3 Port Restricted Cone NAT

The port restricted cone NAT is similar to the restricted cone NAT. However, the port restricted cone NAT also takes the port numbers into account along with the IP addresses. An external host can send a packet with source IP address x and source port p to an internal host only if the internal host has previously sent a packet to the IP address x and port p. The port restricted cone NAT is illustrated in Figure 3.

2.4 Symmetric NAT

In a symmetric NAT, any request from an internal IP address and a port number to some destination IP address and port number is mapped to a unique external IP address and a unique port number. If the same host sends a packet from the same source address and the same port number but to a different destination, a different mapping is used. Only the external host that receives a packet from an internal host can send a UDP packet back to the internal host. The symmetric NAT is illustrated in Figure 4.

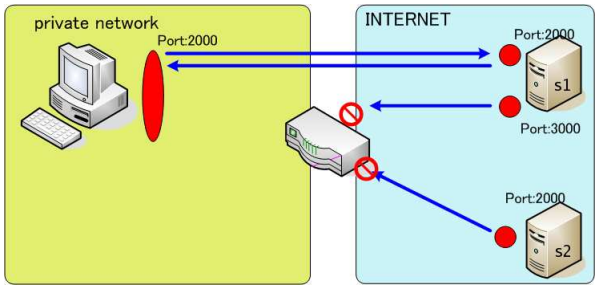

Figure 3: Port restricted cone NAT

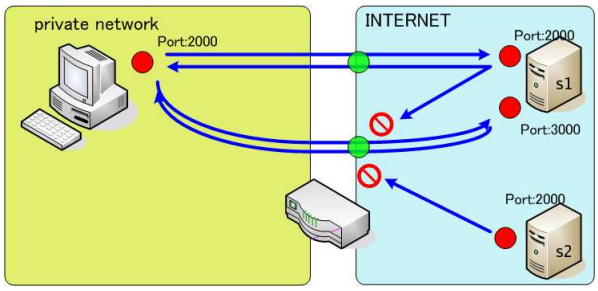

Figure 4: Symmetric NAT

3. EXISTING TRAVERSAL METHODS

There have been many proposals on methods to traverse NATs. This section describes some well-known techniques such as UPnP, STUN, and Teredo. It should be noted here that none of these techniques can be successfully implemented for *symmetric* NATs.

3.1 UPnP

UPnP is a set of computer network protocols promulgated by the UPnP Forum [3]. The UPnP architecture allows the development of *peer-to-peer* networks of PCs, networked appliances, and various wireless devices. When a new host needs a connection, a UPnP device can automatically configure a network address, announce its presence on the network subnet, and exchange a description of device and services. Currently, many Internet gateway vendors such as D-Link, Intel, Buffalo Technology, and Arescom are offering devices with UPnP functionality. NAT traversal in UPnP is known as the Internet Gateway Device (IGD) Protocol. However, one of the disadvantages of UPnP is that it requires that all the devices in the network should support UPnP. Even if a single device does not conform to the UPnP standard, we cannot realize a peer-to-peer network.

3.2 STUN

STUN is a protocol used for communication between a client and a server [4].

If a peer-to-peer software package includes a STUN client, it sends a request to a STUN server. The server then reports back to the STUN client about the global IP address of the NAT router. It also reports the *opened* port number at the NAT for incoming traffic to the private network.

The STUN client determines the type of NAT in use on the basis of the response of the STUN server. There are certain differences in the handling of the incoming of the UDP packets by various types of NATs. STUN works well with three types of NATs: full cone, restricted cone, and port restricted cone NATs. One of the drawbacks of STUN is that it does not work with symmetric NATs which are often used in networks in large enterprises.

In general, STUN assumes that the NAT being used is a *cone* type NAT. If we use a cone NAT, the IP address and the port number of the STUN client are fixed. STUN cannot handle symmetric NATs because the global IP address and port number translated from the private address and port number of the client are *not* fixed.

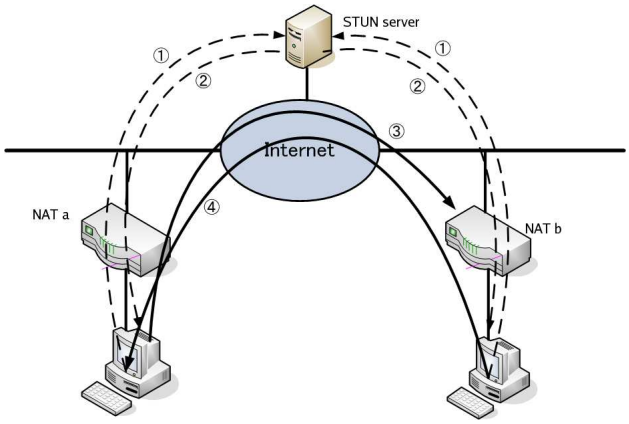

Figure 5: Using STUN to traverse NAT

3.3 Teredo

Teredo has been proposed by Microsoft [6]. It is based on IPv6 tunneling technology. A Teredo client obtains a Teredo IPv6 address from the Teredo server. It utilizes a IPv6 address with IPv6 tunneling in UDP/IPv4. A Teredo client communicates with other Teredo clients and other IPv6 nodes through a Teredo relay. A Teredo server should have both an IPv4 global address and an IPv6 global address. A Teredo relay should have an IPv4 global address and an IPv6 global address. Teredo relay provides routing between the Teredo clients and nodes on the IPv6 Internet.

Teredo does not work well with symmetric NATs.

4. NEW METHOD

In this section we propose a new method for UDP *multi-hole punching*. This method establishes a UDP connection between two end points through NATs, as shown in Figure 6.

The new method is based on port *prediction* and limited TTL values. The method controls the port numbers to allow successful traversing of symmetric NATs. It also works well for the other types of NATs. In addition, the new method can be extended to NAT traversal in TCP. It is well known that NAT traversal in TCP is more difficult than that in UDP. This is an advantage of the proposed new method.

4.1 Phase I

The new method is divided into three phases. In this method, the client is known as an *echo client* and the server is known as an *echo server* because there are a series of packet exchanges between them. An echo client communicates with two servers S1 and S2. S1 and S2 record the IP address and port number of the echo client, and these are then translated by NAT a. The following are the steps of the method:

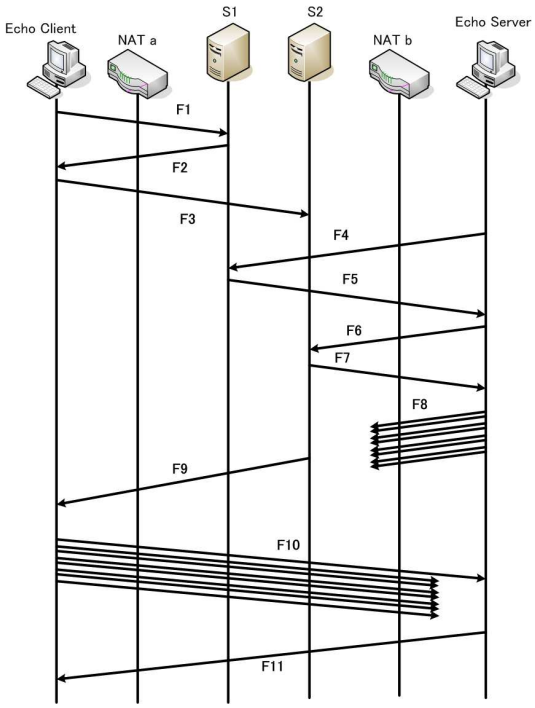

Figure 6: New method of UDP multi-hole punching

F1: The echo client communicates with S1. Then, S1 analyzes the port number mapped by NAT a.

F2: S1 conveys the port number to the echo client.

F3: The echo client sends a packet to S2. It includes information obtained on the port number of NAT a when the echo client communicated with S1. Then, S2 analyzes the port number of NAT a and records it. Furthermore, S2 also records the information obtained on the port number of NAT a when the echo client communicated with S1 at step F1.

4.2 Phase II

In phase II, the echo server communicates with S1 and S2 in a manner similar to that in phase I.

F4: The echo server communicates with S1. Then, S1 analyzes the port number mapped by NAT b.

F5: S1 conveys the port number to the echo server.

F6: The echo server sends a packet to S2. The packet includes the port number information of NAT b obtained from the communication of the echo server with S1 at step F4. Then, S2 analyzes the port number of NAT b and records it. Furthermore, S2 records the port number information of NAT b obtained when the echo server communicated with S1 at step F4.

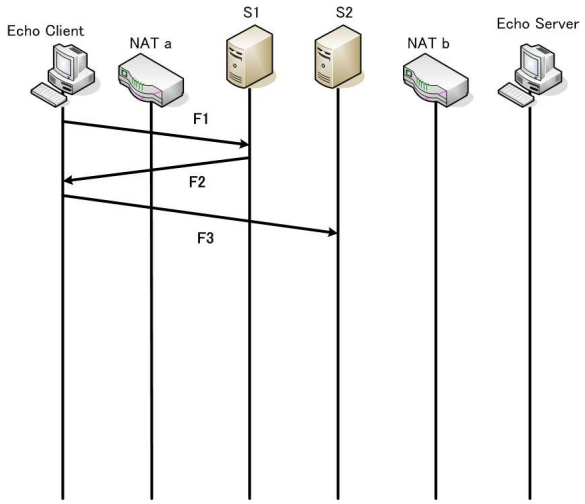

Figure 7: Phase I

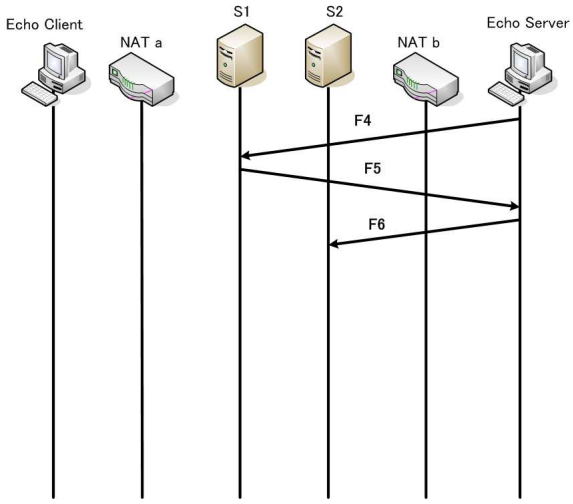

Figure 8: Phase II

4.3 Phase III

In phase III, the method performs *port prediction*. As described in phase I, NAT a maps the port number twice, one each in steps F1 and F3. For example, if NAT a uses 5361 in F1 and 5362 in F3, then we can *predict* that the punching mode of NAT a is incremental and that the predicted port number is 5363. Thus the new method can determine the punching mode as *incremental*, *decremental*, or the *skip* mode. Then, it communicates the target global IP address and the punching mode to the echo client and the echo server. The echo client and the echo server receive the information and then initiate multi-hole punching to establish communication between them.

F7: Based on the two types of information communicated in phases I and II, namely the communications of NAT a with S1 and S2, we can predict a suitable port number for hole punching. We can also determine the punching mode. S2 sends the information containing the predicted port number and the punching mode to the echo server.

F8: Based on this information, the echo server sends a large number of packets. These packets have a fixed destination port and a low TTL value. The echo server binds the port. The packets are then sent to the echo client.

F9: Using the two kinds of information obtained in phases I and II, namely, the communications of NAT b with S1 and S2, we can predict a suitable port number for the hole punching. S2 sends the information that contains the predicted port number and the punching mode to the echo client in a manner similar to that of step F7.

F10: On the basis of the information obtained in step F9, the echo client sends many packets to the echo server. These packets have a fixed destination port. The echo client binds the port. After sending all the packets, the echo client switches to the receiving mode.

In step F10, NAT b receives many UDP packets from the echo client. If one of the source port numbers of the echo client matches the destination port number mapped by NAT b, then NAT b translates the packets and sends it to the echo server successfully. The echo server closes all the opened ports except the ports that have successfully received the packets.

F11: The echo server replies to the echo client. It establishes a P2P connection between the echo client and the echo server at this stage.

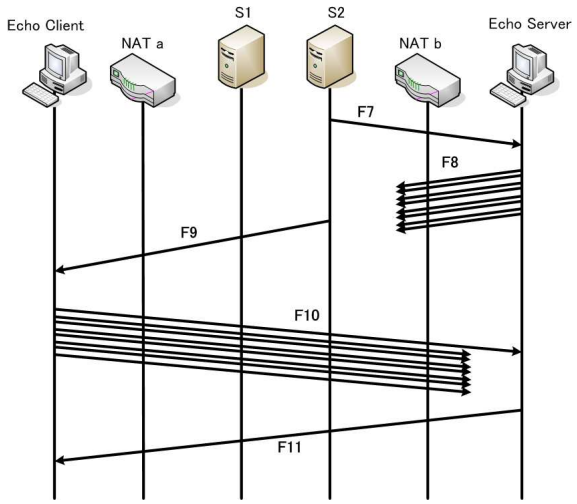

Figure 9: Phase III

4.4 Advantages of the New Method

Normal UDP communications

The new method puts small TTL value, 2 or 3, in a packet from the Echo Server. In step F8 we observe that the packet reaches NAT b but does not reach NAT a. In steps F8 to F11, we can observe that packets go through NAT a or b, as shown in Figure 6 and Figure 9. These packets appears normal UDP communications if we neglect the extra packets.

Thus, UDP packets have less possibility of being discarded because of the use of security criteria for screening packets at NATs.

Precise port number prediction

Most NATs translate port numbers according to one of the following algorithms: *increment*, *decrement*, *leap*, i.e., skipping alternative port numbers, or *random*. Our proposed method uses two servers so that we can observe any type of port translation.

Control of port numbers

The new method uses fixed port numbers rather than random numbers. When the source port numbers are $\{x,\ x+1,\ x+2, \ldots\}$ and the translated port numbers are $\{n,\ n+1,\ n+2, \ldots\}$, we can detect the translation algorithm at a NAT based on the sequences of the source ports and the translated ports. The new method can also detect *random* port translation.

Use of many port numbers

The new method uses many port numbers. The current implementation uses 1,000 port numbers. The large number of ports increases the success rate of hole punching. When NAT a translates port numbers by some unknown algorithm and one of the 1,000 ports matches with the mapping of NAT b, the communication is successfully established.

Stateful Packet Inspection (SPI) in TCP

Currently, many NAT products are equipped with Stateful Packet Inspection (SPI). It is a type of function for filtering of TCP packets. When SPI is applied, a valid sequence of packets should follow the 3-way handshake of TCP. The 3-way handshake is as follows:

1. [SYN] - outgoing
2. [SYN, ACK] - incoming
3. [ACK] - outgoing

The proposed method can be extended to cover TCP NAT traversal. It simulates the 3-way handshake in accordance with SPI criteria. The actual packets are composed by the echo server, the echo client, S1, and S2. A TCP packet has two flags, SYN and ACK, which are easily set. The sequence number is an important field in a TCP packet, which is composed by S1 and S2 for NAT traversal. S1 and S2 monitor the packets from the echo client and echo server to obtain the original sequence numbers. Thus, hole punching can also be extended to TCP using the new traversal method.

5. EXPERIMENTS

We conducted five experiments to evaluate our new method and compared it with other existing methods. We used the WinStun software to classify the type of NATs in the first experiment. In the second experiment we used Wireshark, which is a packet capture tool. In the third experiment, we evaluated the use of the Skype software for NAT traversal. The fourth experiment was performed to test the performance of the new method for UDP NAT traversal. The new method was applied to TCP NAT traversal in the fifth experiment.

We tested several router products, which are shown in Table 1. The first router "Iptables" is not a hardware product but a software tool for routing. It is included in the list so that we can observe the details of Iptables. It is also possible to capture packets at a PC that runs Iptables. The other routers listed are commercial hardware products, and we do not know their detailed internal structure or functions. This is the reason we conducted the first experiment on the classification type of NATs.

Table 1: Routers used in the experiments

manufacturer	model number
Iptables	Iptables
I-O Date	NP-BBRL
I-O Date	WN-WAPG/R
Buffalo	BBR-4HG
Linksys	BEFSR41C-JP3
Planex	BRL-04CW
Corega	CG-BARMX2
NEC	AtermBR1500H
Cisco	Cisco2621

5.1 Classification of Types of NAT using WinStun

We used the WinStun software to classify NATs. WinStun shows two outputs which are the possibility of NAT traversal by STUN and the NAT type. The types of NATs are defined in [7].

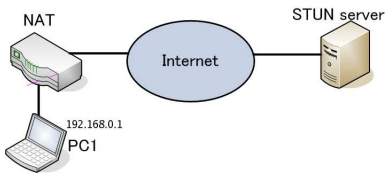

Figure 10: Network configuration of WinStun

The network configuration of the first experiment is shown in Figure 10. The results of the first experiment indicate that Cisco 2621 and IO-Data (WN-WAPG/R) are "impossible to measure". NEC router is "VOIP will NOT work" that means the possibility of symmetric NAT is high. Other routers can also be traversed by STUN. We used nine routes for all the experiments, except the TCP experiment.

5.2 Packet Capturing

In order to study the port translation algorithm, we used a packet capture software. This software monitors the port number information in the packets. The test was conducted for a fixed source port number of 5323. The packets are sent to the appropriate destination port, whose number is changed using the incremental algorithm.

The network configuration of the second experiment is shown in Figure 11. A part of the results is shown in Figure 13, Figure 14, and Figure 12. The results show that the packets are sent to designated port numbers.

The packets were captured at a *switch*, where a mirror function had been enabled.

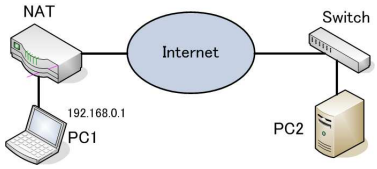

Figure 11: Packet capturing

Protocol	Info
UDP	Source port: 5323 Destination port: 5338
UDP	Source port: 5323 Destination port: 5339
UDP	Source port: 5323 Destination port: 5340
UDP	Source port: 5323 Destination port: 5341
UDP	Source port: 5323 Destination port: 5342
UDP	Source port: 5323 Destination port: 5343
UDP	Source port: 5323 Destination port: 5344
UDP	Source port: 5323 Destination port: 5345
UDP	Source port: 5323 Destination port: 5346
UDP	Source port: 5323 Destination port: 5347
UDP	Source port: 5323 Destination port: 5348

Figure 12: Six routers

Protocol	Info
UDP	Source port: 60286 Destination port: 5333
UDP	Source port: 60287 Destination port: 5334
UDP	Source port: 60288 Destination port: 5335
UDP	Source port: 60289 Destination port: 5336
UDP	Source port: 60290 Destination port: 5337
UDP	Source port: 60291 Destination port: 5338
UDP	Source port: 60292 Destination port: 5339
UDP	Source port: 60293 Destination port: 5340
UDP	Source port: 60294 Destination port: 5341
UDP	Source port: 60295 Destination port: 5342
UDP	Source port: 60296 Destination port: 5343
UDP	Source port: 60297 Destination port: 5344
UDP	Source port: 60298 Destination port: 5345
UDP	Source port: 60299 Destination port: 5346

Figure 13: NEC and I-O Data (WN-WAPG/R)

Protocol	Info
UDP	Source port: 33268 Destination port: 5370
UDP	Source port: 33264 Destination port: 5371
UDP	Source port: 33260 Destination port: 5372
UDP	Source port: 33256 Destination port: 5373
UDP	Source port: 33252 Destination port: 5374
UDP	Source port: 33248 Destination port: 5375
UDP	Source port: 33309 Destination port: 5376
UDP	Source port: 33305 Destination port: 5377
UDP	Source port: 33301 Destination port: 5378
UDP	Source port: 33297 Destination port: 5379
UDP	Source port: 33293 Destination port: 5380

Figure 14: Planex

Figure 12 shows that the source port number of each packet was the same, while Figure 13 and Figure 14 show that three routers (NEC, I-O Date and Planex) translated the source port every time we sent a packet. It should be noted that we sent packets from the fixed source port number 5323. It can observed that the NEC, I-O Data, and Planex routers were of the symmetric NAT type. In addition, the results in Figure 13 show that the translation algorithms of NEC and IO-Data were of the incremental type, while that of Planex was of the random type.

5.3 Use of Skype Software for NAT traversal

Although no formal documentation is available, it appears that Skype applies STUN for P2P communications [8]. For our reference, we tested the Skype P2P connection.

Skype is known to use three types of communication methods — P2P, UDP-RELAY, and TCP-RELAY. We observed the communication methods of Skype by using the *analysis window* of Skype, which is shown in Figure 15. Figure 15 is an example of the analyze window when we use Iptables and an NEC router.

Figure 15: Analysis window of Skype

The results are listed in Table 2, where RU implies Relay UDP, RT is Relay TCP, and "o" is a P2P connected. The oblique sign "/" indicates that there is no need for testing. Table 2 shows the different combinations of the nine routers. The success ratio of the P2P communication was 46%. It is observed that Skype did not use UDP hole punching when the voice quality was good.

5.4 Performance of new method for UDP

We tested the performance of the new method for NAT traversal. Figure 16 illustrates the network configuration for testing. S1 and S2 had global IP addresses. PC1 and PC2 indicated the echo client and echo server, respectively. PC1 and PC2 had private IP addresses. We used a packet capturing tool, Wireshark, at PC1, PC2, and Iptables. Figure 17 shows the captured packets at NAT *b* when we used Iptables. Figure 18 shows the captured data. The complete results are shown in Table. 2. The combination of Buffalo and NEC had an 80% success rate on average. The other combinations were 100% successful.

Table 2: Skype software for NAT traversal

	Iptables	I-O	I-O2	Buffalo	Linksys	Planex	Corega	NEC	Cisco
Iptables	RT	RT	RU	o	o	RT	RT	RU	RT
I-O Data (NP-BBRL)	/	/	o	o	o	RU	o	RU	o
I-O Data (WN-WAPG/R)	/	/	/	o	o	RU	RT	RU	RT
Buffalo	/	/	/	/	RU	RT	o	o	o
Linksys	/	/	/	/	/	o	o	o	o
Planex	/	/	/	/	/	/	RT	RT	RT
Corega	/	/	/	/	/	/	/	RU	o
NEC	/	/	/	/	/	/	/	/	RT
Cisco	/	/	/	/	/	/	/	/	/

Table 3: New method in UDP

	Iptables	I-O	I-O2	Buffalo	Linksys	Planex	Corega	NEC	Cisco
Iptables	o	o	o	o	o	o	o	o	o
I-O Data (NP-BBRL)	/	/	o	o	o	o	o	o	o
I-O Data (WN-WAPG/R)	/	/	/	o	o	o	o	o	o
Buffalo	/	/	/	/	o	o	o	80%	o
Linksys	/	/	/	/	/	o	o	o	o
Planex	/	/	/	/	/	/	o	o	o
Corega	/	/	/	/	/	/	/	o	o
NEC	/	/	/	/	/	/	/	/	o
Cisco	/	/	/	/	/	/	/	/	/

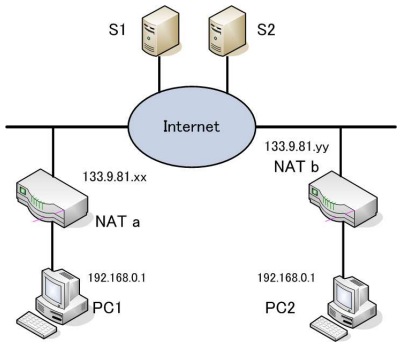

Figure 16: Network configuration for testing

Time	Source	Destination	Protocol	Info
0.000000	133.9.81.62	133.9.81.66	UDP	Source port: 5320 Destination port: 5320
0.000233	133.9.81.66	133.9.81.62	UDP	Source port: 5320 Destination port: 5320
0.000265	133.9.81.66	192.168.0.1	UDP	Source port: 5320 Destination port: 5320
0.000743	192.168.0.1	133.9.81.63	UDP	Source port: 5321 Destination port: 5320
0.003997	133.9.81.62	133.9.81.63	UDP	Source port: 5321 Destination port: 5320
0.004606	133.9.81.63	133.9.81.62	UDP	Source port: 5320 Destination port: 5321
0.004638	133.9.81.63	192.168.0.1	UDP	Source port: 5320 Destination port: 5321
0.004992	192.168.0.1	133.9.81.186	UDP	Source port: 5322 Destination port: 50313
0.005035	133.9.81.62	133.9.81.186	UDP	Source port: 5322 Destination port: 50313
0.005041	192.168.0.1	133.9.81.186	UDP	Source port: 5323 Destination port: 50314
0.005052	133.9.81.62	133.9.81.186	UDP	Source port: 5323 Destination port: 50314
0.012234	133.9.81.126	133.9.81.62	ICMP	Time-to-live exceeded (Time to live exceede
0.012280	133.9.81.126	192.168.0.1	ICMP	Time-to-live exceeded (Time to live exceede
0.012287	133.9.81.126	133.9.81.62	ICMP	Time-to-live exceeded (Time to live exceede
0.012295	133.9.81.126	192.168.0.1	ICMP	Time-to-live exceeded (Time to live exceede
0.012350	133.9.81.126	133.9.81.62	ICMP	Time-to-live exceeded (Time to live exceede
2.014061	133.9.81.186	133.9.81.62	UDP	Source port: 50313 Destination port: 5322
2.014108	133.9.81.186	192.168.0.1	UDP	Source port: 50313 Destination port: 5322
2.019802	192.168.0.1	133.9.81.186	UDP	Source port: 5322 Destination port: 50313
2.019831	133.9.81.62	133.9.81.186	UDP	Source port: 5322 Destination port: 50313
2.032795	133.9.81.186	133.9.81.62	UDP	Source port: 50314 Destination port: 5623

Figure 17: Data captured at start of traversal

Source	Destination	Protocc	Info
133.9.81.186	133.9.81.62	UDP	Source port: 5361 Destination port: 5361
133.9.81.186	133.9.81.62	UDP	Source port: 5362 Destination port: 5362
133.9.81.186	133.9.81.62	UDP	Source port: 5363 Destination port: 5363
133.9.81.186	133.9.81.62	UDP	Source port: 5364 Destination port: 5364
133.9.81.186	133.9.81.62	UDP	Source port: 5365 Destination port: 5365
133.9.81.186	133.9.81.62	UDP	Source port: 5366 Destination port: 5366
133.9.81.186	133.9.81.62	UDP	Source port: 5367 Destination port: 5367

Figure 18: Captured data of Iptables and Planex

5.4.1 *Detailed analysis*

Figure 17 shows a record of the sequence of packets when the echo client and echo server start communication through NATs. Since the packets are captured at NAT *b*, the record does not cover steps F1, F2, and F3, which do not go through NAT *b*. The "Time" field indicates time stamps represented by a second.

The packets at the instants 0.000000 and 0.004638 correspond to steps F4, F5, F6, and F7. S1 has an IP address 192.9.81.66. The address of NAT *b* is 133.9.81.62, S2 is 133.9.81.63, and the echo server (PC2) is 192.168.0.1. From the instant 0.004992 to 0.005052, the echo server (=PC2) sent packets with a low TTL via NAT *b*. This was in step F8. The packets, whose ICMP times were exceeded, returned between the time period from 0.012234 to 0.012350. At the instant 2.014061, the echo server received a packet from the echo client via NAT *a* at 133.9.81.186. This was in step F10. The packet was forwarded to the echo server (=PC2) at time 2.014108. The port number 5322 successfully sent the packet. At instant 2.019802 and 2.019831, the echo server send a packet to the echo client using the port number 5322. This was in step F11 and it indicated successful NAT traversal. The last line of Figure 17 shows a packet at port 5623, which was an extra packet because the port 5322 was already known and used for NAT traversal.

5.4.2 Control of port numbers

From Figure 18, we can see that Iptables has an IP address of 133.9.81.186, while Planex is at 133.9.81.62. It is observed that the source port numbers are incremental: 5361, 5362, ... In Figure 14, Planex translates the port numbers *randomly*. The new method can rectify the random port number using the *incremental* algorithm, as observed in Figure 18

5.5 The new method in TCP

We tested the performance of the new method for traversing NATs in TCP. The extended method is based on UDP hole punching. It uses UDP communication to exchange information over a TCP connection. It then simulates the TCP 3-way handshake sequence and generates a sequence of TCP segments with proper sequence numbers.

The results show that the new method can successfully traverse NATs in TCP for five out of six products. We have one failure because the specific product has a filtering function on port numbers as well as SPI. It allows a local port is designated to unique destination at {IP address, port} only. It is more restrictive than SPI.

6. CONCLUSION

This paper proposes a new method of hole punching in order to traverse NATs successfully. The method utilizes two servers to predict port numbers. The new method sends UDP packets with a low TTL value. The experiments show successful results for the NAT products available in the market.

Table 4: Summary of NAT traversal in UDP

	WinStun	Skype	New method
Symmetric NAT	33%	0%	100%
All NATs	66%	46%	97%

Our method has the following advantages:

- It assigns a low TTL value to the packets. An ICMP *time exceeded* message is generated when the TTL value becomes zero. It should be noted that NAT ports are still open when TTL = 0.
- The method successfully deals with the *random* port translation algorithm by controlling the fixed port numbers and observing the behavior of NATs.
- The success rate of the NAT traversal with Skype software is 46%. Our new method outperforms it with a success rate of 99%. This method can be used symmetric NAT products such as NEC, I-O Data (WN-WAPG/R), and Planex routers.
- STUN does not succeed in traversing the NATs of NEC, I-O Data (WN-WAPG/R), or the Cisco routers. The new method can be successfully used even for those NATs.

Since the new method needs two servers, it increases the cost of infrastructure. The high success rate can justify the overhead cost in the proposed method.

The new method can also be used for TCP hole punching, which is more difficult than UDP hole punching if we follow the existing methods. We have succeeded in establishing TCP connections for five NAT products out of six. The proposed method outperforms the existing methods significantly.

7. REFERENCES

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, SIP: Session Initiation Protocol, RFC 3261, June 2002. http://www.ietf.org/rfc/rfc3261.txt

[2] http://www.packetizer.com/ipmc/h323/

[3] UPnP Forum, http://www.upnp.org/

[4] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs), RFC 3489, March 2003. http://tools.ietf.org/html/rfc3489

[5] L. Daigle, Ed., IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation, RFC 3424, November 2002. http://www.ietf.org/rfc/rfc3424.txt

[6] Teredo and IPv6 support, http://www.ipv6style.jp/en/tryout/20030929/index.shtml

[7] F. Audet, Ed., C. Jennings, NAT Behavioral Requirements for Unicast UDP draft-ietf-behave-nat-udp-08, October 10, 2006. http://tools.ietf.org/wg/behave/draft-ietf-behave-nat-udp/draft-ietf-behave-nat-udp-08.txt

[8] S. A. Baset and H. Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, September 15, 2004. http://arxiv.org/pdf/cs/0412017

[9] W. R. Stevens, TCP/IP Illustrated Volume 1: The Protocols, Addison-Wesley, 1994.

[10] Y. Murayama, TCP/IP Network programing, Ohmsha, Tokyo, 2002.

[11] E. R. Harold, Java Network Programming, O'Reilly Japan, 2001.

[12] S. Ikejima, The mechanism about Skype, Nikkei BP, 2005.

[13] S. Guha, and P. Francis, Characterization and Measurement of TCP Traversal through NATs and Firewalls, Oct. 2005. http://nutss.gforge.cis.cornell.edu/pub/imc05-tcpnat.pdf

[14] Y. Takeda, Internet draft: Symmetric NAT Traversal using STUN, June 2003. Work in progress. http://www.cs.cornell.edu/projects/stunt/draft-takeda-symmetric-nat-traversal-00.txt

[15] P. Srisuresh, and M. Holdrege, IP Network Address Translator (NAT) Terminology and Considerations, RFC 2663, August 1999. http://www.faqs.org/rfcs/rfc2663.html

[16] Tomo's HotLine http://homepage3.nifty.com/toremoro/index.html

[17] Yahoo! PtoP live broadcast (Cnet) http://japan.cnet.com/news/media/story/0,2000056023,20249687,00.htm