# Tunneling IPv6 through NAT with Teredo Mechanism

**Shiang-Ming Huang, Quincy Wu, Yi-Bing Lin**
*Department of Computer Science and Information Engineering*
*National Chiao Tung University*
*{smhuang,solomon,liny}@csie.nctu.edu.tw*

## Abstract

*Teredo is a service that enables hosts located behind one or more IPv4 NATs to obtain IPv6 connectivity by tunneling packets over IPv4 UDP. Under the national IPv6 deployment project in Taiwan, we developed the first Linux-based Teredo service in 2003. In this paper, we explain how IPv6 candidates located behind NATs can enlist the help of "Teredo servers" and "Teredo relays" to learn their "global addresses" and to obtain connectivity, and how clients, servers and relays can be organized in Teredo networks. We also describe in details our strategies for implementing Teredo server and Teredo relay under Linux, and show the performance of different Teredo implementations in public domain.*

**Keywords: IPv6, Teredo, Tunneling, NAT.**

## 1. Introduction

After three decades of evolution in Internet technologies, IETF (Internet Engineering Task Force) has developed Internet Protocol version 6 (IPv6) as the next generation Internet protocol. In comparison with the existing Internet Protocol version 4 (IPv4), IPv6 provides larger address space, more efficient routing mechanism, better support to security mechanism and quality of service (QoS).

During the IPv6 deployment stage, many existing IP networks remain to support IPv4 only. Because each site may deploy IPv6 in a highly diffuse and incremental fashion, it is unrealistic to assume a single Flag Day to upgrade all IPv4 networks to IPv6. To facilitate the transition for IPv4 to IPv6 migration, *tunneling* techniques are utilized as a routing infrastructure to carry IPv6 traffic through IPv4 networks [1].

In existing IPv6-in-IPv4 tunneling mechanisms such as configured tunnel & automatic tunnel [1], 6to4 tunnel [2] and tunnel broker [3], both end points of a tunnel must possess public IPv4 addresses. Although public IPv4 addresses may be available in normal scenarios, many Internet service providers, especially WLAN (wireless local area network) and GPRS (general packet radio service), usually provide private IPv4 addresses to their customers, and NAT (network address translation) [4] is utilized to establish Internet connectivity. Hence, IPv6 users behind the NAT would be unable to establish tunnels to other IPv6 networks. This turns out to be one of the major obstacles in IPv6 deployment.

Several solutions for tunneling IPv6 packets through NAT have been proposed, including VPN (virtual private network) and UDP tunnel [5]. These solutions provide IPv6 connectivity for private hosts. However, they have the scalability problem because manual configuration is required for the user end of a tunnel. This configuration is not an easy task for common users, and is not suitable for ISP to perform large deployment. Moreover, in these approaches, only one static tunnel server is assigned to relay all IPv6 packets of the private network. This tunnel server may potentially become the bottleneck, and it is very likely that the traffic follows a "dog leg" route from the source to the tunnel server and then to the destination that is not an optimal routing path. Recently, an enhanced model of UDP tunnel called "Silkroad" [6] was also proposed to alleviate the bottleneck issue. However, some critical algorithms are still missing in the draft to have a thorough investigation on its performance. To solve the above issues, Teredo [7] was proposed as an automatic tunneling mechanism that is capable of traversing NATs.

Under the support of NICI (National Information and Communication Initiative), we developed the first Linux-based Teredo to speed up the IPv6 deployment in Taiwan. In this paper, we shall introduce the Teredo mechanism that helps tunneling IPv6 through NAT, and then describe our implementation of Teredo and the performance measurement.

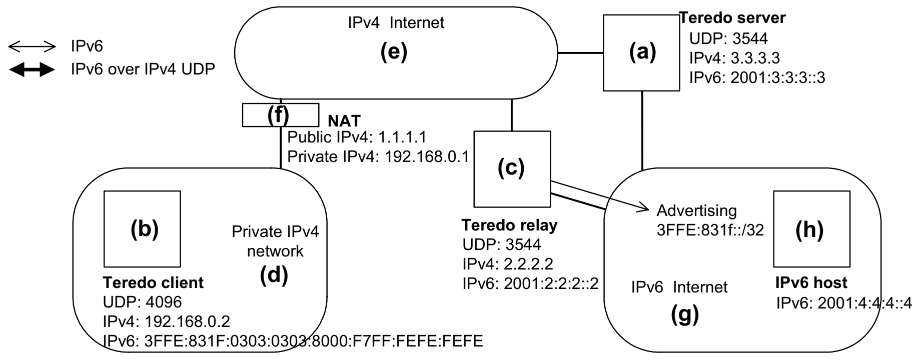

**Figure 1. Teredo architecture**

## 2. The Teredo solution

Teredo provides IPv6 connectivity for IPv6 hosts on private IPv4 networks with NAT by tunneling IPv6 packets over IPv4 UDP. The Teredo architecture is illustrated in Figure 1, which consists of a Teredo server (Figure 1 (a)), several Teredo clients (Figure 1 (b)) and Teredo relays (Figure 1 (c)). A Teredo client provides a Teredo tunnel interface for an IPv6 host on a private IPv4 network (Figure 1 (d)) that connects to IPv4 Internet (Figure 1 (e)) with NAT (Figure 1 (f)). A Teredo server assists a Teredo client to obtain IPv6 address for IPv6 Internet access. The Teredo server is stateless and only has to handle a small volume of traffic. A Teredo relay encapsulates IPv6 packets addressed to Teredo clients in IPv4 UDP, and decapsulates IPv4 UDP packets sent from the Teredo client to IPv6. Every Teredo relay advertises a 3FFE:831F::/32 IPv6 address prefix to IPv6 Internet (Figure 1 (g)). With the advertisement, all IPv6 packets sent from different IPv6 hosts (Figure 1 (h)) to a Teredo client are routed to Teredo relays closest to the packet sources, and therefore traffic load can be dynamically shared among Teredo relays.

The NAT translates the private IPv4 address of all pass-through packets according to an address mapping table. Suppose that the NAT is a "full cone" NAT [8], its address mapping table consists of the following fields (see Figure 2).

| private IPv4 address | private port | public IPv4 address | public port |
|---|---|---|---|

**Figure 2. Address mapping table format**

As an automatic tunneling mechanism, Teredo embeds the NAT traversal information in its 128-bit IPv6 address. A Teredo IPv6 address format consists of the following fields (see Figure 3). The "3FFE:831F::/32" field specifies the IPv6 address prefix of Teredo. The "Teredo server IPv4 address" field indicates the IPv4 address of a Teredo server. The **"**Flag" filed indicates the type of NAT ("full cone" or not) [8]. The **"**Obfuscated mapped public UDP port" and the "Obfuscated mapped public IPv4 address" fields indicate the public trasnport address on NAT that is mapped from the Teredo client's private transport address.

Formatted: Bullets and Numbering

The necessity for the obfuscation mechanism is that, some "smart NAT" scans payload of pass-through IPv4 packets and translates any 32-bit substring in the payload that matches the address to be translated in the IPv4 header (or translates any 16-bit substring in the payload that matches the port to be translated in the TCP/UDP header). Therefore, to prevent these "smart NATs" from modifying the Teredo IPv6 addresses in the encapsulated IPv4 UDP packets, obfuscation performs bitwise XOR operation on the original value with 1 to "protect" it.

A Teredo client obtains a Teredo IPv6 address from the Teredo server after performing a *qualification* procedure. This procedure detects the NAT type and informs the Teredo client about the public mapped transport address. As long as Teredo clients get Teredo IPv6 addresses, they are able to communicate with IPv6 Internet with the help of Teredo servers and Teredo relays. The details are described in an example below.

| 3FFE:831F::/32 | Teredo server IPv4 address | Flag | Obfuscated mapped public UDP port | Obfuscated mapped public IPv4 address |
|---|---|---|---|---|
| 32 bits | 32 bits | 16 bits | 16 bits | 32 bits |

**Figure 3. Teredo IPv6 address format**

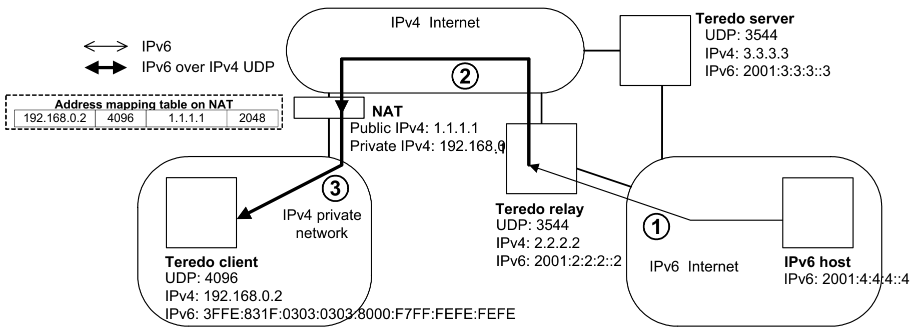

**Figure 4. Communication between a Teredo client and an IPv6 host**

Communication between a Teredo client and an IPv6 host is illustrated in Figure 4. In this example, the IPv4 address of the Teredo server is 3.3.3.3. The NAT has two interfaces: the WAN interface has public IPv4 address 1.1.1.1, while the LAN interface has private IPv4 address 192.168.0.1. The Teredo client has IPv4 address 192.168.0.2. Suppose that the Teredo client uses UDP port 4096 to request a Teredo address from the Teredo Server. When this packet arrives at the NAT, the NAT dynamically allocates an available UDP port (2048 in this example) for this connection, and creates an entry in the mapping table with private transport address 192.168.0.2:4096 and public transport address 1.1.1.1:2048.

The Teredo server calculates the Teredo IPv6 address of this client by determining the values of each field:

- □ Prefix (32 bits) = 0x3FFE831F
- □ Teredo server IPv4 address (32 bits) = 3.3.3.3 = 0x03030303
- □ Flag (16 bits) = 0x8000 ("full cone" NAT)
- □ Obfuscated mapped public UDP port (16 bits) = 2048 □0xFFFF = 0x0800□0xFFFF = 0xF7FF
- □ Obfuscated mapped public IPv4 address (32 bits) = 1.1.1.1 □ 0xFFFFFFFF = 0x01010101 □ 0xFFFFFFFF = 0xFEFEFEFE

Therefore, the Teredo IPv6 address assigned by the Teredo Server to this Teredo client is 3FFE:831F:0303:0303:8000:F7FF:FEFE:FEFE.

Suppose that an IPv6 packet is delivered from an IPv6 host to a Teredo client. The detailed steps are described as follows.

**Step 1.** The IPv6 packet is sent from the IPv6 host to the Teredo relay.

**Step 2.** The Teredo relay encapsulates the IPv6 packet in IPv4 UDP. The IPv4 address information and UDP port information of this packet are determined based on Teredo IPv6 address as follows. The source IPv4 address is the IPv4 address of the Teredo relay (2.2.2.2), and the destination IPv4 address is the "original value" derived from the "Obfuscated mapped public IPv4 address" field of destination IPv6 address (0xFEFEFEFE □ 0xFFFFFFFF = 0x01010101 = 1.1.1.1). The source UDP port is 3544, and the destination UDP port is the "original value" derived from the "Obfuscated mapped public UDP port" field of destination IPv6 address (0xF7FF□ 0xFFFF = 0x0800 = 2048). The Teredo relay sends the IPv4 UDP packet to the NAT (1.1.1.1) through IPv4 Internet.

**Step 3.** After the NAT receives this IPv4 UDP packet, it translates the destination IPv4 address and destination UDP port from 1.1.1.1:2048 to 192.168.0.2:4096 according to the address mapping table, and then sends it to the Teredo client.

Upon reception of the packet, the Teredo client decapsulates the IPv4 UDP packet and obtains the IPv6 packet sent from the IPv6 host.

For scenarios of traversing other types of NATs (restricted cone and port-restricted cone) and scenarios of packets transmission from Teredo clients to public IPv6 hosts, please refer to the Teredo draft [7] for details.

## 2.1. Linux-based Teredo

This subsection describes a Teredo implementation on Linux, namely, NICI-Teredo [9]. During its development in 2003, an independent implementation for FreeBSD was also developed by 6WIND [10], and in 2004, Miredo-Teredo was also developed on Solaris, FreeBSD, and Linux [11].

NICI-Teredo supports Teredo server and Teredo relay functions that can be installed on a single host or independently on distributed hosts. The Teredo server function is developed in a user level daemon. A user level daemon is sufficient since a Teredo server is stateless with small volume of traffic to handle. The Teredo relay function is developed with a combination of a user level program and a kernel level module. A user level program deals with NICI-Teredo relay configuration, while a kernel level module provides a high speed IPv6 packet relaying function. Detailed software architectures of NICI-Teredo are described as follows.

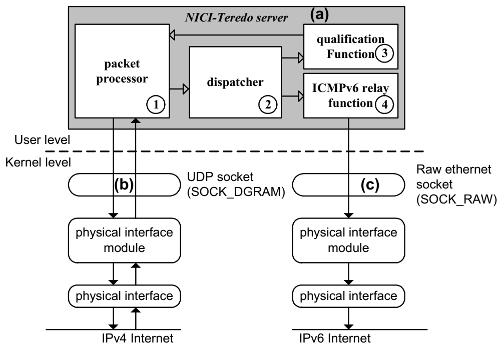

**Figure 5. Architecture of NICI-Teredo server**

NICI-Teredo server (Figure 5 (a)) creates two IPv4 Internet connected UDP sockets (Figure 5 (b)) and an IPv6 Internet connected raw ethernet socket (Figure 5 (c)) on execution. These sockets allow this daemon to receive and transmit IPv6 packets. NICI-Teredo server consists of four components. The "packet processor" (Figure 5 ①) handles IPv6 packets encapsulation and IPv4 UDP packets decapsulation. The "dispatcher" (Figure 5 ②) checks the IPv6 packets delivered from the "packet processor" and dispatches them to the proper functions. The "qualification function" (Figure 5 ③) provides capability to perform qualification procedure for Teredo clients. It helps the Teredo client to discover the type of NAT and the mappped public transport address. The "ICMPv6 relay function" serves as a message relay for Teredo clients to find a closest Teredo relay. Detailed operations of these functions can be found in the draft and are not elaborated in this paper.

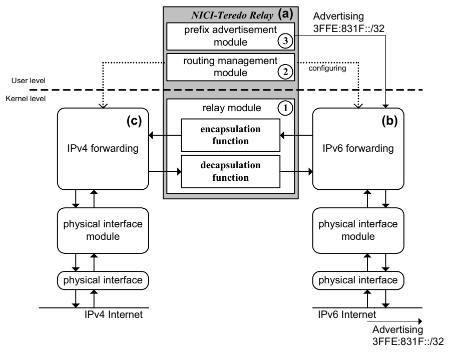

**Figure 6. Architecture of NICI-Teredo relay**

NICI-Teredo relay (Figure 6 (a)) provides the IPv6 packet relay function with the help of IPv6 and IPv4 forwarding (Figure 6 (b) and (c)) mechanisms in Linux kernel. This Teredo relay consists of three modules. The "relay module" (Figure 6 ①) provides IPv6 packets encapsulation and decapsulation functions. It encapsulates the IPv6 packets forwarded from IPv6 routing mechanism in IPv4 UDP, and then passes them to IPv4 routing mechanism where they will be delivered to the Teredo clients. The encapsulated IPv6 packets are also handled by this module. It decapsulates these packets to IPv6, and then passes them to IPv6 routing mechanism where they will be delivered to normal IPv6 hosts. The "routing management module" (Figure 6 ②) manages the packet forwarding plans in kernel. It configures the IPv6 forwarding plan to forward IPv6 packets which are destined to Teredo clients (destination IPv6 address matches 3FFE:831F::/32 prefix) to the "relay module", and configures the IPv4 forwarding plan to forward IPv4 UDP packets sent from Teredo clients to the "relay module". The "prefix advertisement module" (Figure 6 ③) advertises the 3FFE:831F::/32 IPv6 address prefix to IPv6 Internet. This advertisement helps IPv6 packets destined to Teredo clients being routed to the closest Teredo relay in IPv6 networks.

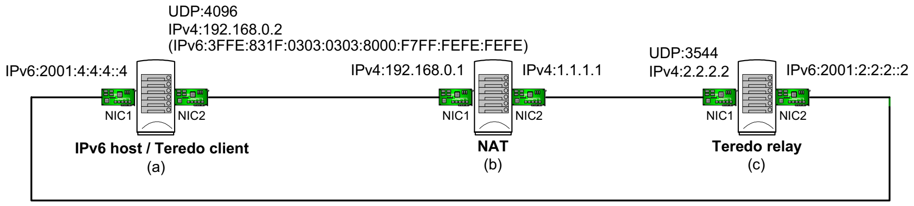

**Figure 7. Measurement environment**

In comparison with other Linux-based Teredo relay [10,11], the packet relay function of NICI-Teredo relay is developed in a kernel level module while that of Miredo-Teredo relay is developed in a user level program. The packet processing latency of NICI-Teredo relay is thus shorter because there are no packet copying operations between kernel level and user level. This will be elaborated in next section. Moreover, NICI-Teredo relay provides a simple way for installation though it involves kernel hacking. Since this module is dynamic loadable, the installation does not need to modify or re-compile the Linux kernel.

## 3. Performance measurement

Teredo relay is the bottleneck component in Teredo mechanism which will handle large volume of network traffic. In this section, we measure the performance of different implementations of Teredo relays. The output measurement is "packet processing latency", which includes both packet encapsulation latency (from a public IPv6 host to a private Teredo client) and packet decapsulation latency (from a private Teredo client to a public IPv6 host).

The measurement environment consists of three hosts (see Figure 7). This environment follows the Teredo architecture in Figure 1. According to the testing architecture in RFC 2544 [12], we use a tester (Figure 7 (a)) which configures both the Teredo client function and the IPv6 host on it. This IPv6 host runs on Redhat Linux 9 with an IPv4 UDP daemon to simulate the Teredo client function. The NAT (Figure 7 (b)) runs on Redhat Linux 9 with address mapping rule set by *iptables*. The DUT (device under test) is the Teredo relay (Figure 7 (c)) in three different versions. It runs on Redhat Linux 9 for NICI-Teredo relay measurement and Miredo-Teredo relay measurement, and runs on FreeBSD 4.9 for 6WIND-Teredo relay measurement. The hardware specification of these hosts is the same, which is a personal computer (PC) with 1800+ AMD Athlon CPU, 256 MB SDRAM and two RealTek 8139 100BaseTx Ethernet cards.

In the measurement, a C program with pcap library [13] is used for catching the packet receiving and delivering timestamps. Pcap is a packet capturing library supported on multiple operating systems. The tests follow the packet sending rule as the round-trip time measurement utility *ping*. That is, we send one packet per second to the Teredo relay for encapsulation or decapsulation, and measure the packet processing latency. Tests are conducted with three different packet sizes (64 bytes, 512 bytes, 1280 bytes), respectively. Each test generates 10,000 packets.

Figure 8 shows the packet encapsulation latency histograms of the three Teredo relays (payload size is 1280 bytes, the IPv6 MTU of Teredo). The latency of the NICI-Teredo relay clusters around 7µs~9µs and 11µs~13µs; the latency of the 6WIND-Teredo relay is around 11µs~15µs; the latency of Miredo-Teredo relay clusters is around 28µs~32µs and 37µs~40µs. The 1280-byte packet decapsulation latency histogram is similar to Figure 8, and the average packets processing latency of the three Teredo relays are listed in Table 1.

Formatted: Bullets and Numbering

With different packet sizes, the encapsulation and decapsulation latency histograms of NICI-Teredo relay and 6WIND-Teredo relay are similar to the histogram in Figure 8, while those of Miredo-Teredo relay takes longer delay when the packet size grows bigger.

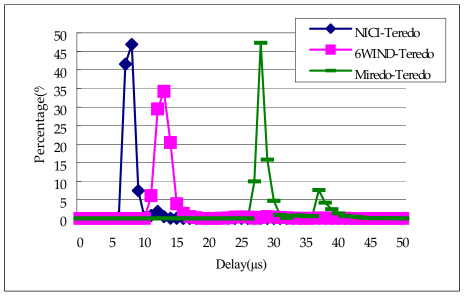

**Figure 8. Encapsulation latency histograms of Teredo relay (1280 bytes)**

**Table 1. Average processing latency**

| Teredo relay | Avg. latency of encapsulation (µs) | Avg. latency of decapsulation (µs) |
|---|---|---|
| NICI | 7.82 | 9.03 |
| 6WIND | 13.53 | 14.63 |
| Miredo | 30.46 | 33.14 |

## 4. Conclusions & Future Work

This paper introduces the NAT traversal problem in IPv6 deployment process in Taiwan. We developed NICI-Teredo with a pretty good performance to solve the NAT problem. NICI-Teredo is also the first Teredo implementation on Linux. As a NAT traversable automatic tunneling mechanism, Teredo greatly simplifies the IPv6 deployment process and facilitates the IPv6 transition stage.

Although Teredo is a useful mechanism, it does not work for all kinds of NAT devices. According to the rules for port mapping and access control, there are different types of NATs. As described in RFC 3489, the four major types are full cone NAT, restricted cone NAT, port restricted cone NAT, and symmetric NAT. Although Teredo can successfully traverse the former three types of NATs, it fails in traversing symmetric NATs. Whether it is possible to enhance the protocol design of Teredo so that it could traverse symmetric NATs, is still an open problem for further research.

## Acknowledgement

This project is partially sponsored by NICI IPv6 R&D Division, Taiwan.

## References

[1] R. Gilligan and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000.

[2] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001.

[3] A. Durand, P. Fasano, I. Guardini and D. Lento, "IPv6 Tunnel Broker", RFC 3053, January 2001.

[4] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.

[5] H. Levkowetz and S. Vaarala, "Mobile IP Traversal of Network Address Translation (NAT) Devices", RFC 3519, April 2003.

[6] M. Liu , X. Wu, Y. Cai, M. Jin and D. Li, "Tunneling IPv6 with private IPv4 addresses through NAT devices", Internet Draft, draft-liumin-v6ops-silkroad-01.txt (Work In Progress), May 2004.

[7] C. Huitema, "Teredo: Tunneling IPv6 over UDP through NATs", Internet draft, draft-huitema-v6ops-teredo-02.txt (Work In Progress), March 2004.

[8] 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

[9] S.M. Huang and Q. Wu, "Implementation of Teredo – Tunneling IPv6 through NATs", Technical Report for National Information and Communication Initiative (NICI) IPv6 R&D Division, Republic of China, 2003.

[10] 6WIND-Teredo, http://www-rp.lip6.fr/teredo/

[11] Miredo-Teredo, http://www.simphalempin.com/dev/miredo/

[12] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, March 1999.

[13] Pcap library, http://www.tcpdump.org/

**Formatted:** Bullets and Numbering