WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2006; **6**:585–599
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/wcm.412

SIP mobility and IPv4/IPv6 dual-stack supports in 3G IP multimedia subsystem

Shiang-Ming Huang[1], Quincy Wu[2*,†], Yi-Bing Lin[1] and Che-Hua Yeh[1]

[1]*Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan*
[2]*Graduate Institute of Communication Engineering, National Chi Nan University, Nantou 545, Taiwan*

Summary

In the Universal Mobile Telecommunications System (UMTS), session initiation protocol (SIP) and IPv6 are the default protocols for IP multimedia core network subsystem (IMS). However, a user equipment (UE) may not be allowed to roam or hand off from UMTS to a private-IPv4 GPRS network. In this paper, we utilize SIP mobility and an automatic IPv6 tunneling mechanism, called Teredo, to support roaming/handoff of a UE between different networks. We have developed the first non-commercial Linux-based Teredo mechanism, and compared our solution with other Teredo implementations in the public domain. Our study indicates that our solution can reduce the tunneling overhead and transmission delay over two other implementations by 44–74%. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: IPv6; NAT; SIP terminal mobility; Teredo; tunneling; UMTS

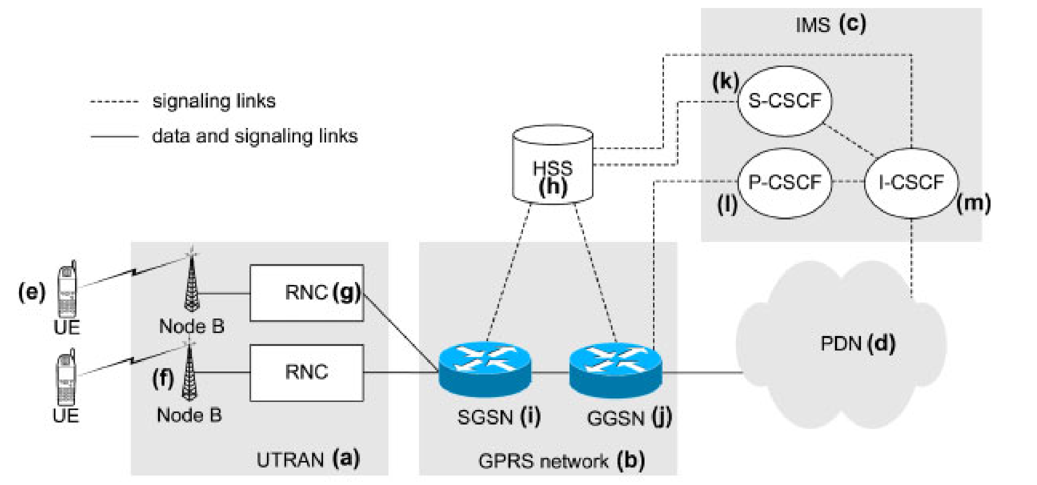
1. Introduction

Universal Mobile Telecommunications System (UMTS) is a third-generation (3G) mobile network proposed by the Third Generation Partnership Project (3GPP). UMTS effectively integrates Internet protocol (IP) with cellular technologies. Specifically, the IP multimedia core network subsystem (IMS) utilizes IP version 6 (IPv6) to support large IP address space, and provides a variety of multimedia services based on session initiation protocol (SIP) [1].

Figure 1 illustrates the UMTS architecture for packet-switched service domain [2–5]. The general packet radio service (GPRS) network (Figure 1(b)) is an IP-based backbone network consists of serving GPRS support nodes (SGSNs; Figure 1(i)) and gateway GPRS support nodes (GGSNs; Figure 1(j)). An SGSN relays IP packets between the mobile users and the GGSN. A GGSN is a specialized router that functions like a gateway. It controls user data sessions and transfers the data packets between the GPRS network and the external packet data network (PDN; Figure 1(d)). The home subscriber server (HSS; Figure 1(h)) is a master database containing all 3G user-related subscription information. Both the GPRS and the IMS networks (Figure 1(c)) access the HSS for mobility management and session management. The UMTS terrestrial radio access network (UTRAN; Figure 1(a)) consists

*Correspondence to: Quincy Wu, Graduate Institute of Communication Engineering, National Chi Nan University, No. 1, University Road, Puli, Nantou 545, Taiwan.
†E-mail: Quincy.Wu@Gmail.com

Contract/grant sponsor: National Science Council; contract/grant numbers: NSC93-2219-E-009-029; NSC93-2752-E-0090005-PAE.



CSCF: Call session control function
GGSN: Gateway GPRS support node
GPRS: General Packet Radio Service
HSS: Home subscriber server
I-CSCF: Interrogating CSCF
IMS: IP multimedia core network subsystem
Node B: Base station
P-CSCF: Proxy CSCF
PDN: Packet data network
RNC: Radio network controller
S-CSCF: Serving CSCF
SGSN: Serving GPRS support node
UE: User equipment
UMTS: Universal Mobile Telecommunications System
UTRAN: UMTS terrestrial radio access network

Fig. 1. The UMTS network architecture.

of Node Bs (the UTRAN base stations; Figure 1(f)) and radio network controllers (RNCs; Figure 1(g)) connected by an asynchronous transfer mode (ATM) network. The user equipment (UE; Figure 1(e)) is a mobile phone with an IP protocol stack. A UE communicates with a Node B through radio interface based on wideband code division multiple access (WCDMA) radio technology.

The UEs utilize the GPRS network to access the IMS network. A packet data protocol (PDP) context specifies a connection between the UE and the GGSN, and the UE needs to activate the PDP context before it proceeds to access the IMS services. From the PDP context activation procedure, the UE obtains an IP address assigned by the GGSN. Note that the GGSN-assigned IP address can be in the IPv4 (IP version 4) or in the IPv6 format [6].

The IMS is a SIP-based service infrastructure where IMS signaling is exercised at proxy call session control function (P-CSCF; Figure 1(l)), interrogating CSCF (I-CSCF; Figure 1(m)) and serving CSCF (S-CSCF; Figure 1(k)). The P-CSCF is the first contact point within the IMS for a UE. It acts as a SIP proxy server between the UE and I-CSCF/S-CSCF. The I-CSCF is responsible for routing SIP messages from P-CSCF to an appropriate S-CSCF, and the S-CSCF provides actual SIP service functions. Some of these CSCFs may only support IPv6 (e.g., 3GPP Release 5) and thus cannot be accessed from an IPv4-only network.

During IPv6 deployment, many existing IP networks remain to support IPv4 only, and a UE may find itself attached to a GPRS network that does not support IPv6. Because each network may deploy IPv6 in 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 in UMTS, *tunneling* techniques are utilized to carry IPv6 traffic through IPv4 networks [7]. A dual-stack UE (a UE with both IPv4 and IPv6 protocol stacks) can utilize the GGSN-assigned IPv4 address to establish a tunnel for connection with external IPv6 networks [8]. We will address this transition issue later.

In existing IPv6-in-IPv4 tunneling mechanisms such as configured tunnel and automatic tunnel [7], 6 to 4 tunnel [9] and tunnel broker [10], both end points of a tunnel must possess public IPv4 addresses. Although public IPv4 addresses may be available in some typical scenarios, many Internet service providers, especially WLAN (wireless local area network) and GPRS [5], may only provide private IPv4 addresses to their customers, and NAT (network address translation) [11] is required to establish Internet connectivity. Hence, IPv6 users within private IPv4 networks would not be able to establish tunnels to IPv6 networks. This issue turns out to be one of the major obstacles in the deployment of IPv6 environment (e.g., UMTS IMS).

Several IPv6 tunneling solutions for private IPv4 networks with NAT have been proposed, including virtual private network (VPN) and user datagram

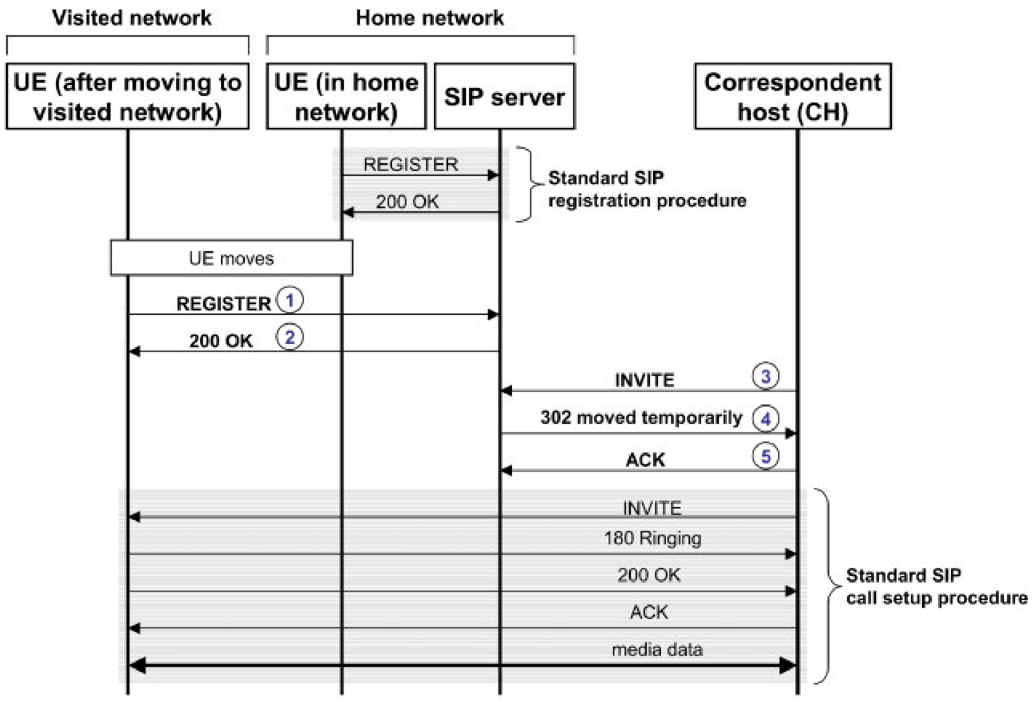


Fig. 2. SIP pre-call mobility procedure.

protocol (UDP) tunnel [12]. These solutions provide IPv6 connectivity for private hosts, but require manual configuration at the user end of a tunnel. This configuration task is not transparent to users, and is not easy for novice users. Therefore, these solutions are not suitable for large private IPv4 networks. Moreover, in these approaches, only one static tunnel server is assigned to relay all IPv6 packets of a host in 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, resulting in a non-optimal routing path. To address the above issues, the Internet society proposes Teredo [13], an automatic tunneling mechanism with the capability to traverse NATs.[‡]

Another important issue in UMTS is mobility support for UEs. Within a single GPRS network, the UE mobility is supported by the SGSN and the GGSN with a layer-2 protocol called GPRS tunneling protocol (GTP). However, if a UE hands off to different visited networks with different IPv4/IPv6 support (e.g., a UE hands off to different operator's IP network), the situation becomes complicated. In this paper, we combine an application level mechanism, called SIP terminal mobility, and an automatic IPv6 tunneling mechanism, called Teredo, to support the handoff of a UE between different networks, including private IPv4 networks as well as IPv6 networks.

This paper is organized as follows. Section 2 describes the mobility support with SIP in UMTS network. Specifically, we elaborate on the procedures for a UE to access the IMS services when it hands off from its home network (UMTS) to a visited network (UMTS or GPRS). Section 3 describes the Teredo mechanism that enables a UE to access the IPv6-only IMS via a private IPv4 PDP context. In Section 4, we investigate the design of our Linux-based Teredo software architecture, and then show the performance comparison with other public-domain Teredo solutions.

2. SIP Terminal Mobility

SIP is an application-layer signaling protocol for establishing, modifying, and terminating multimedia sessions [1]. The SIP protocol is capable of handling terminal mobility, session mobility, personal mobility, and service mobility [15]. Here, we focus on SIP terminal

[‡] Recently, an enhanced model of UDP tunnel called "Silkroad" [14] was proposed to alleviate the bottleneck issue. However, some critical algorithms of Silkroad are still missing at the time when this paper is written.

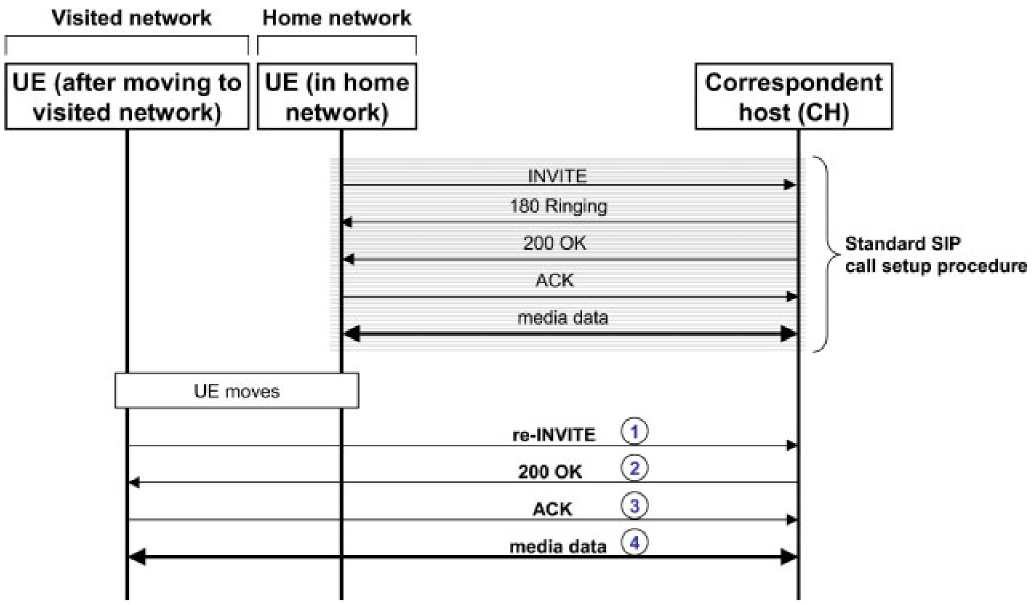


Fig. 3. SIP mid-call mobility procedure.

mobility that allows a mobile host (i.e., a UE in UMTS) to move between IP subnets while remaining reachable to correspondent hosts (CHs). The SIP terminal mobility comprises *SIP pre-call mobility* (Figure 2) and *SIP mid-call mobility* (Figure 3), in which the former enables a CH to establish sessions to a UE, and the latter enables a UE to re-establish its on-going sessions while it moves to a visited network. In UMTS network, these two actions correspond to *roaming* and *handoff*, respectively.

The SIP pre-call mobility is illustrated in Figure 2. In this figure, a UE moves from its home network (UMTS) to a visited network (GPRS or UMTS), and initiates the SIP pre-call mobility procedure as described below.

Steps A.1 and A.2. The UE sends a SIP REGISTER request to update its new contact address with the SIP server.

Steps A.3 and A.4. When a CH sends a SIP INVITE request to the UE through the SIP server, the SIP server notifies the CH of the UE's new contact address.

Step A.5. The CH replies with a SIP ACK message to notify the SIP Server that it has received the SIP 200 OK response.

Upon receipt of the UE's new contact address, the CH contacts this address directly and a session is established between the CH and the UE through the standard SIP call setup procedure.

In addition to SIP pre-call mobility which is applied to allow call establishment of successive sessions after the UE moves, for UE movement in the middle of a session, SIP mid-call mobility must be applied to re-establish the on-going sessions. After the UE moves to a visited network and obtains a new contact address, it modifies the existing session by issuing a new SIP INVITE request, in which the SIP header fields 'From', 'To', 'Call-ID' have identical values as those in the original SIP INVITE request which establishes this session. This secondary SIP INVITE request is called a SIP re-INVITE request, and it modifies an existing session by new parameters specified in the 'connection address' field of the session description protocol (SDP) and new 'Contact' field in the SIP header.

The SIP mid-call mobility is illustrated in Figure 2, where the UE and the CH first establish a session through the standard SIP call setup procedure, and then the UE moves to a visited network, and re-establishes this session with the following steps.

Step B.1. When the UE moves, it initiates the SIP mid-call mobility mechanism by sending a SIP re-INVITE request to the CH. In this request, the contact field in the SIP header and the SDP connection address field are updated to the UE's new IP address.

Step B.2. When the CH receives this request, it replies a SIP 200 OK response.

Step B.3. The UE replies with an SIP ACK message to notify the CH that it has received the SIP 200 OK response.

Step B.4. The CH modifies the session parameters according to the new connection address in the SDP content, and then the media data transmission is re-established between the CH and the UE with its new address.

The format of a SIP re-INVITE request is identical to a SIP INVITE request. Thus it is unnecessary to modify the SIP protocol or create a new SIP method. Moreover, SIP terminal mobility can fit the requirements of fast handoff, low latency, and high bandwidth utilization [15,16].

The above examples assume that both the UE and the CH utilize IPv6 as the underlying protocol. If the visited network is a private IPv4-only network (e.g., some GPRS networks), the UE must first obtain IPv6 connectivity before it proceeds to utilize SIP terminal mobility mechanism. In Reference [8] an extended mechanism based on mobile IPv6 was proposed to support the handoff of a mobile node from its IPv6 home network to any private IPv4-only networks. However, in 3GPP IMS, SIP terminal mobility is believed to be a better mobility solution than mobile IPv6, because SIP is the native protocol supported in the IMS. We shall illustrate how to adopt SIP terminal mobility to support UE roaming and handoff, and utilize Teredo mechanism to provide IPv6 connectivity.

The Teredo SIP terminal mobility solution is illustrated in Figure 4. In this figure, the UE and the CH are the two communication ends as described in the previous examples. The UE hands off from an IPv6 UMTS network to a private IPv4-only GPRS network with an NAT. The UE utilizes Teredo and SIP terminal mobility mechanisms with the following steps:

Steps C.1 and C.2. The UE hands off to a private IPv4-only GPRS network. It utilizes the Teredo mechanism to obtain IPv6 connectivity (details will be elaborated in Section 3).

Step C.3. The UE and the CH utilize the Teredo mechanism to send and receive IPv6 packets. They perform the SIP terminal mobility procedure as described in **Steps A.1–A.5** and **Steps B.1–B.4**, for pre-call and mid-call scenarios, respectively.

Step C.4. After execution of the SIP terminal mobility procedure, the UE and the CH can communicate successfully with the help of IPv6 packet relay function provided by Teredo.

Details of the Teredo mechanism will be elaborated in the following section.

3. Teredo mechanism

By tunneling IPv6 packets over IPv4 UDP through NATs, Teredo provides IPv6 connectivity for dual-stack nodes within private IPv4 networks. The Teredo-supported UEs, therefore, are able to access the IPv6-only IMS from a private IPv4-only GPRS network.

The Teredo architecture is illustrated in Figure 5, which consists of a Teredo server (Figure 5(a)), several Teredo clients Figure 5(b)) and Teredo relays Figure 5(c)). A Teredo client is implemented at a dual-stack node in a private IPv4 network (i.e., a GPRS

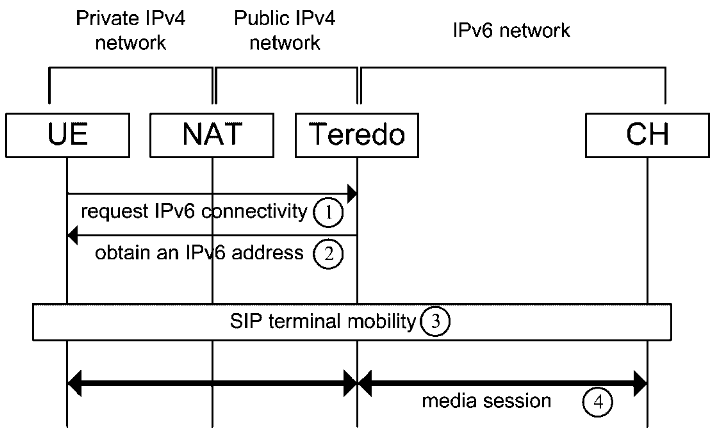


Fig. 4. The Teredo SIP terminal mobility solution.

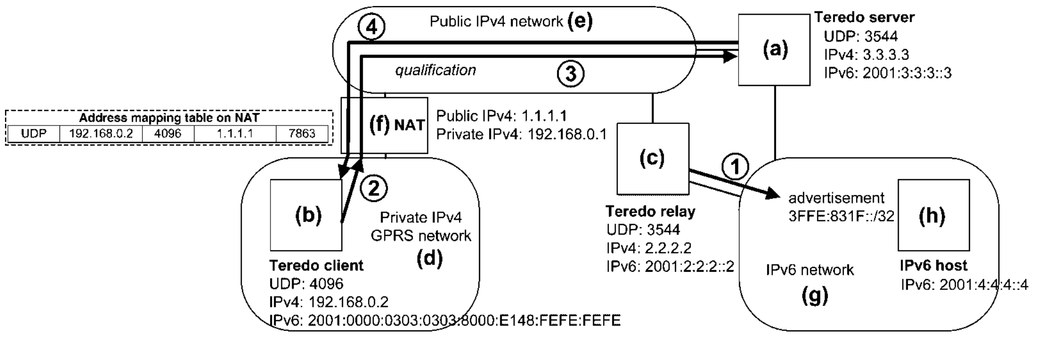


Fig. 5. Teredo architecture.

transport protocol	private IPv4 address	private port	public IPv4 address	public port

Fig. 6. An entry in the address mapping table.

network in Figure 5(d)) that connects to public IPv4 network Figure 5(e)) through NAT Figure 5(f)). A Teredo server assists a Teredo client to obtain an IPv6 address for IPv6 network access. A Teredo relay establishes IPv4 UDP tunnels with the Teredo clients using the designated port 3544, and relays IPv6 packet between the Teredo clients and the IPv6 network. For IPv6 packets sent from the IPv6 network and destined to a Teredo client, the Teredo relay encapsulates these packets in IPv4 UDP and forwards them to the destination Teredo client in a private IPv4 network. In the reverse direction, the Teredo relay decapsulates IPv4 UDP packets sent from the Teredo client to the IPv6 network Figure 5(g)). To broadcast its identity, every Teredo relay advertises an IPv6 address prefix 2001:0000::/32 to the IPv6 network (path ① in Figure 5). Through the advertisement, the IPv6 hosts (Figure 5(h)) select appropriate Teredo relays such that all IPv6 packets sent from these IPv6 hosts to a Teredo client are routed to Teredo relays closest to the packet sources. Therefore the traffic load to a Teredo client can be dynamically adjusted among Teredo relays with optimal routing.

The Teredo client in Figure 5 represents a dual-stack UE with Teredo client software running on it; the NAT is most likely to be implemented at the GGSN, and thus the public IPv4 network is the PDN connected to the GGSN.

In Figure 5, the IPv4 address of the Teredo server is 3.3.3.3. The NAT is equipped with two network interfaces: the WAN interface (to the public network) has the public IPv4 address 1.1.1.1, and the LAN interface (to the private network) has the private IPv4 address 192.168.0.1. The Teredo client is assigned the private IPv4 address 192.168.0.2. The NAT performs private–public address translation for all pass-through packets according to an address-mapping table. Suppose that the NAT is a full cone NAT[§] [17]. The fields of an entry in the NAT address-mapping table are illustrated in Figure 6. In this table, the 'private IPv4 address' and the 'private port' fields store the private transport address of the private end (i.e., IPv4 address plus TCP/UDP port, whose values are 192.168.0.2 and 4096 for the example in Figure 5). The 'transport protocol' field stores the transport protocol type (TCP or UDP). The 'public IPv4 address' and the 'public port' fields store the public transport address assigned by the NAT. The values in these fields (IPv4 address 1.1.1.1 and port 7863 in this example) are used to replace the private transport address. This public transport address of a private IPv4 host is crucial for translating IPv4 UDP packets passing through the NAT. When a host in the public IPv4 network sends an IPv4 UDP packet to this transport address (1.1.1.1:7863), the NAT dispatches this packet to the

[§] When an internal host within a private network sends a packet to an external host, an NAT maps the private transport address of the internal host to a unique public transport address. Thereafter, a public host can send packets to the private host by delivering them to the mapped public transport address. However, different types of NATs have different rules to handle incoming packets. The full cone NAT allows packets from all public hosts to pass through, while a restricted cone NAT allows an external host (with IP address X) sending a packet to the private transport address only if the private transport address had previously sent a packet to IP address X.

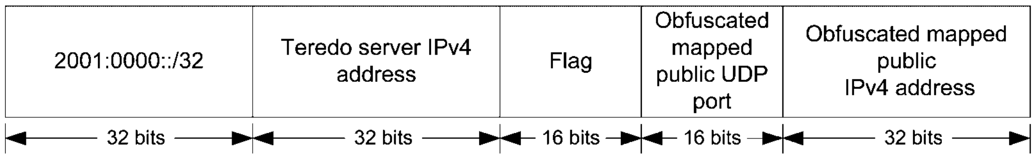

2001:0000::/32
Teredo server IPv4 address
Flag
Obfuscated mapped public UDP port
Obfuscated mapped public IPv4 address
32 bits
32 bits
16 bits
16 bits
32 bits

Fig. 7. Teredo IPv6 address format.

designated private IPv4 host (192.168.0.2:4096) with the help of the address-mapping table.

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 fields illustrated in Figure 7. The '2001:0000::/32' field specifies the IPv6 address prefix for Teredo. The 'Teredo server IPv4 address' field indicates the IPv4 address of a Teredo server (e.g., 3.3.3.3 in Figure 5). The 'Flag' field indicates the type of NAT (full cone or not) [17]. The 'Obfuscated mapped public UDP port' and the 'Obfuscated mapped public IPv4 address' fields indicate the public transport address assigned by the NAT. This transport address is mapped to the Teredo client's private transport address. The obfuscation mechanism is needed because some NAT products provide 'generic' application layer gateway (ALG) functionality. The generic ALG hunts for IPv4 addresses, either in text or binary formats within a packet, and rewrites them if they match a binding in the address-mapping table. When this 'smart NAT' handles the payload of pass-through IPv4 packets, it translates any occurrence of IPv4 address in the payload that matches the address to be translated in the IPv4 header (or translates any occurrence of port number in the payload that matches the port to be translated in the TCP/UDP header). Such action certainly interferes with normal Teredo operations. To prevent the smart NAT 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.

At start-up, a Teredo client (i.e., a UE) obtains a Teredo IPv6 address by performing the *qualification* procedure with the Teredo server (path ② → ③ → ④ in Figure 5). From this procedure, the Teredo client detects the NAT type and learns its mapped public transport address from the Teredo server. As long as the Teredo clients obtain Teredo IPv6 addresses, they are able to communicate with the IPv6 network with the help of Teredo servers and Teredo relays. In the example of Figure 5, the Teredo client uses UDP port 4096 to initiate the qualification procedure with the Teredo server. When this UDP request arrives at the NAT (path ② in Figure 5), the NAT dynamically allocates an available UDP port (e.g., 7863 in this example) for this connection, and creates an entry in the address mapping table with protocol type UDP, private transport address 192.168.0.2:4096 and public transport address 1.1.1.1:7863. Then the UDP request is sent to the Teredo server (path ③ in Figure 5), and the Teredo server replies a UDP response containing the public transport address (e.g., 1.1.1.1:7863 in this example) observed by the Teredo server (path ④ in Figure 5). The Teredo client repeats this process following the NAT type detection algorithm in Reference [17], and then calculates its Teredo IPv6 address by determining the value of each field as follows:

- Prefix (32 bits) = 0x20010000
- 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) = 7863 $\oplus$ 0xFFFF = 0x1EB7 $\oplus$ 0xFFFF = 0xE148
- Obfuscated mapped public IPv4 address (32 bits) = 1.1.1.1 $\oplus$ 0xFFFFFFFF = 0x01010101 $\oplus$ 0xFFFFFFFF = 0xFEFEFEFE

Therefore, the Teredo IPv6 address obtained by this Teredo client is 2001:0000:0303:0303:8000:E148:FEFE:FEFE.

After qualification, communication between the Teredo client and an IPv6 host can be established. Figure 8 illustrates how an IPv6 packet is delivered from an IPv6 host to a Teredo client with the following steps (see path ① → ② → ③ in Figure 8).

Step D.1. The IPv6 packet is sent from the IPv6 host to the Teredo relay. (The IPv6 host selects this Teredo relay according to the IPv6 address prefix 2001:0000::/32 advertisement; see path ① in Figure 5.)

Step D.2. The Teredo relay encapsulates the IPv6 packet in an IPv4 UDP packet. The source address is the IPv4 address of the Teredo relay (2.2.2.2), and the source UDP port is 3544. The destination IPv4 address and UDP port of this packet are determined based on the IPv6 address of this Teredo client as follows. The

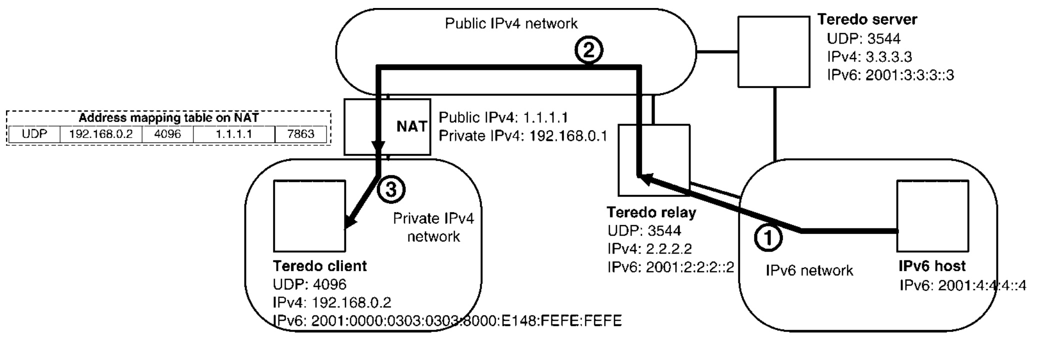


Fig. 8. Communication between a Teredo client and an IPv6 host.

destination address is the restored value derived from the 32-bit 'Obfuscated mapped public IPv4 address' field of destination IPv6 address (0xFEFEFEFE $\oplus$ 0xFFFFFFFF = 0x01010101 = 1.1.1.1), and the destination UDP port is the restored value derived from the 16-bit 'Obfuscated mapped public UDP port' field of destination IPv6 address (0xE148 $\oplus$ 0xFFFF = 0x1EB7 = 7863). The Teredo relay sends the IPv4 UDP packet to the NAT (1.1.1.1:7863) through the IPv4 network.

Step D.3. When the NAT receives this IPv4 UDP packet, it translates the destination IPv4 address and UDP port from 1.1.1.1:7863 to 192.168.0.2:4096 according to the address mapping table, and then sends it to the Teredo client in the private IPv4 network.

Upon receipt of the packet, the Teredo client decapsulates the IPv4 UDP packet to obtain the IPv6 packet. With the above steps, IPv6 packets sent from the IPv6 host can successfully pass through the NAT. For IPv6 packet delivery from a Teredo client to an IPv6 host, please refer to the Teredo specification [13]. The above example assumes a full cone NAT server. Details of packet transmission for other types of NATs can also be found in the Teredo specification [13].

4. Three Teredo Implementations

We implemented NICI-Teredo [18] on Linux in year 2003. In the same year, an independent implementation for FreeBSD was also developed by 6WIND, LIP6 and Euronetlab [19]. In year 2004, Miredo-Teredo was developed on Solaris, FreeBSD, and Linux [20]. In this section, we illustrate the software architectures of these Teredo implementations.

NICI-Teredo supports the Teredo server and Teredo relay functions that can be installed on a single host or independently on multiple hosts. The Teredo server function is implemented as a user-level daemon, which is illustrated in Figure 9 (a). The Teredo relay function is developed with a combination of a user-level program and a kernel-level module as illustrated in Figure 10 (a). The user-level program deals with the NICI-Teredo configuration, while the kernel-level module supports high-speed IPv6 packet relaying.

The NICI-Teredo server (Figure 9(a)) invokes an IPv6 raw Ethernet socket (Figure 9(b)) and two IPv4 UDP sockets (Figure 9(c)) at the Linux kernel to send and receive packets. The NICI-Teredo server consists of four components. The *packet processor* (Figure 9 ①) handles IPv6 packet encapsulation and IPv4 UDP packet decapsulation. The *dispatcher* (Figure 9 ②) checks the IPv6 packets delivered from the *packet processor* and dispatches them to the proper functions. The *qualification function* (Figure 9 ③) performs the qualification procedure for Teredo clients. This function helps the Teredo client to discover the type of NAT and the mapped public transport address, as described in Section 3. The Teredo server interacts with the Teredo clients for qualification through path Ⓐ → Ⓑ → Ⓒ → Ⓓ → Ⓔ in Figure 9. For a Teredo client that attempts to communicate with an IPv6 host, the *ICMPv6 relay function* (Figure 9 ④) helps to locate a Teredo relay closest to the destination IPv6 host. This discovery task is achieved by the ICMPv6 echo request sent from the Teredo client to the destination IPv6 host through path Ⓐ → Ⓑ → Ⓕ → Ⓖ in Figure 9 and the response sent from the IPv6 host to the Teredo client following path ① → ② → ③ in Figure 8. From the response message, the Teredo relay is located for the communication between the Teredo client and the IPv6 host. As the other two implementations of Teredo servers, both 6WIND and Miredo have similar architecture designs as NICI-Teredo server, and the details are omitted.

The NICI-Teredo relay (Figure 10(a)) provides the IPv6 packet relay function by utilizing the IPv6 and IPv4 forwarding mechanisms (Figure 10(b) and (c)) at

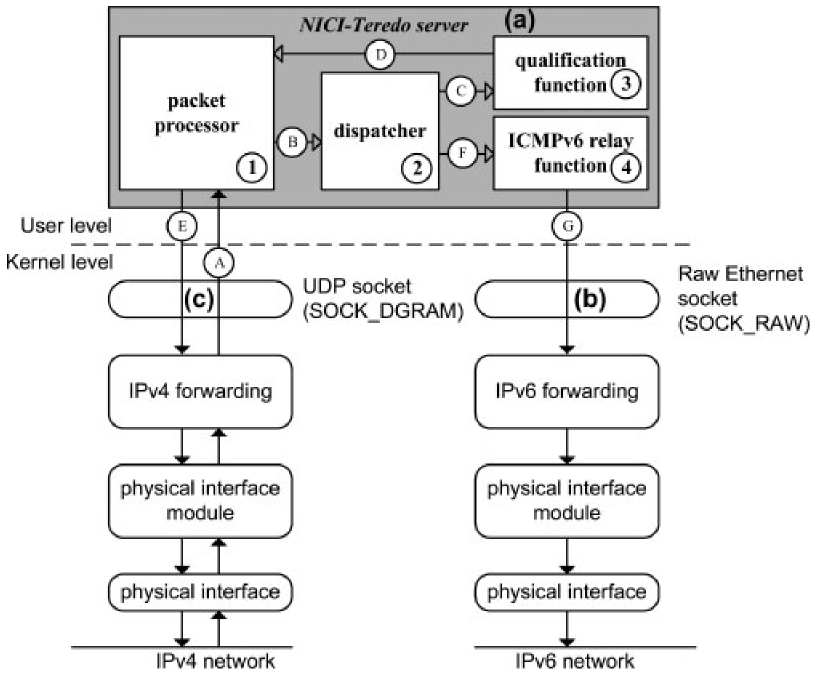


Fig. 9. Software architecture of the NICI-Teredo server.

the Linux kernel. This Teredo relay consists of three modules. The *prefix advertisement module* (Figure 10 ①) advertises the IPv6 address prefix 2001:0000::/32 to the IPv6 network so that the IPv6 packets destined to the Teredo clients can be routed to the nearest Teredo relay. The *routing management module* (Figure 10 ②) initializes the packet forwarding plans at the Linux kernel. It configures the IPv6 forwarding plan to route the IPv6 packets to the Teredo clients (i.e., for the packets with the destination IPv6 addresses matching the IPv6 address prefix 2001:0000::/32) through the *relay module*, and configures the IPv4 forwarding plan to dispatch the IPv4 UDP packets from the Teredo clients (i.e., for the packets with destination port matching the designated port 3544) to the *relay module*. The *relay module* (Figure 10 ③) provides packet encapsulation

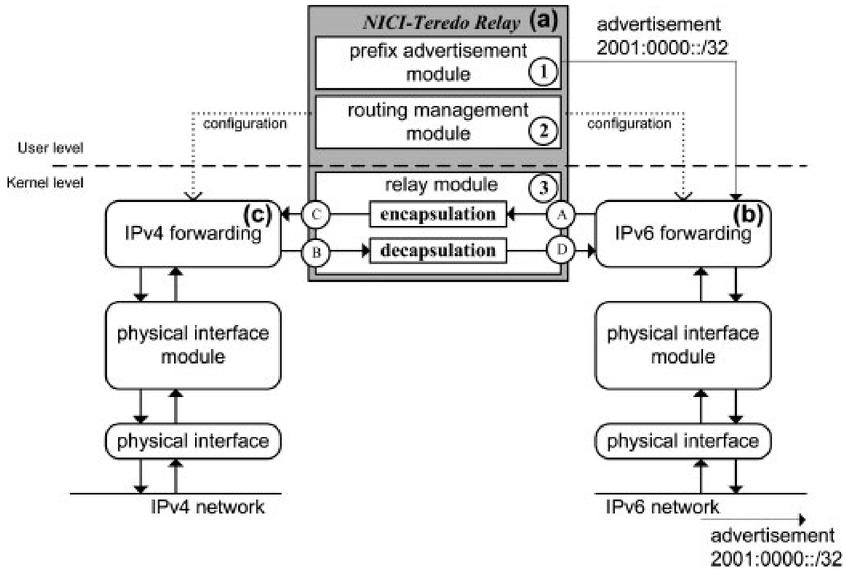


Fig. 10. Software architecture of the NICI-Teredo relay.

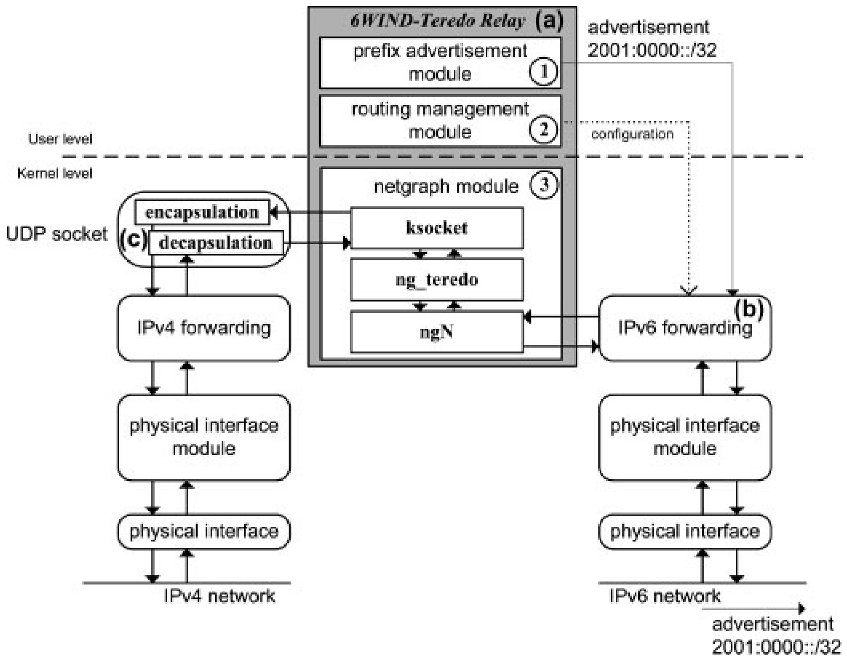


Fig. 11. Software architecture of the 6WIND-Teredo relay.

and decapsulation functions by two callback functions, `udpip6_tunnel_xmit()` (Figure 10 Ⓐ) and `udpip6_rcv()` (Figure 10 Ⓑ). These functions are invoked by the IPv6 and the IPv4 forwarding mechanisms, respectively. From the IPv6 forwarding mechanism, the *relay module* receives IPv6 packets for IPv4 UDP encapsulation, and then passes them to the IPv4 forwarding mechanism by `IPTUNNEL_XMIT`[‖] (Figure 10 Ⓒ). The encapsulated packets are then delivered to the Teredo clients. In the reverse direction, the tunneled IPv4 UDP packets are decapsulated into IPv6 by this module, and then passed to the IPv6 forwarding mechanism (i.e., via the Linux function `netif_rx()`) for delivery to the IPv6 networks (Figure 10 Ⓓ).

The 6WIND-Teredo relay (Figure 11(a)) utilizes *netgraph*, a FreeBSD in-kernel networking subsystem, and the IPv6 forwarding mechanism (Figure 11(b)) and socket functions (Figure 11(c)) at the FreeBSD kernel to provide packet relay function. Unlike the NICI-Teredo relay that utilizes a single kernel-level module to provide all packet-processing functions, the 6WIND-Teredo relay relies on the cooperation of several *netgraph* submodules and a kernel IPv4 UDP socket to handle the packets. The 6WIND-Teredo relay consists of three components. The *prefix advertisement module* (Figure 11 ①) advertises the IPv6 address prefix 2001:0000::/32 to the IPv6 network, which provides similar functions as the prefix advertisement module in the NICI-Teredo relay. The *routing management module* (Figure 11 ②) initializes the IPv6 forwarding plan to route the IPv6 packets to the Teredo clients through the *netgraph module*. The *netgraph module* (Figure 11 ③) utilizes three submodules to deal with IPv6 packet processing. The `netisr_dispatch()` and `ng_iface_output()` functions of the *ngN* submodule connect to the IPv6 protocol stack to deliver IPv6 packets; the `so_pru_sosend()` and `so_pru_soreceive()` functions of the *ksocket* submodule utilize a kernel IPv4 UDP socket with port 3544 to deliver the encapsulated packets with the Teredo clients, where the IPv4 UDP socket provides packet encapsulation and decapsulation functions. These two *netgraph* submodules interact with each other through the *ng_teredo* submodule. Specifically, the *ng_teredo* submodule uses `NG_SEND_DATA()` and `ng_teredo_rcvdata()` to deliver IPv6 packets between the *ngN* and the *ksocket* submodules. Through cooperation of these three submodules in the *netgraph module*, IPv6 packets received by the *ngN* submodule are written to the kernel IPv4 UDP socket by the *ksocket* submodule, and vice versa.

[‖] According to the Linux programming convention, the uppercase name `IPTUNNEL_XMIT()` refers to a macro, while the lowercase name such as `udpip6_rcv()` refers to a function. The same convention is used in the 6WIND implementation on FreeBSD.

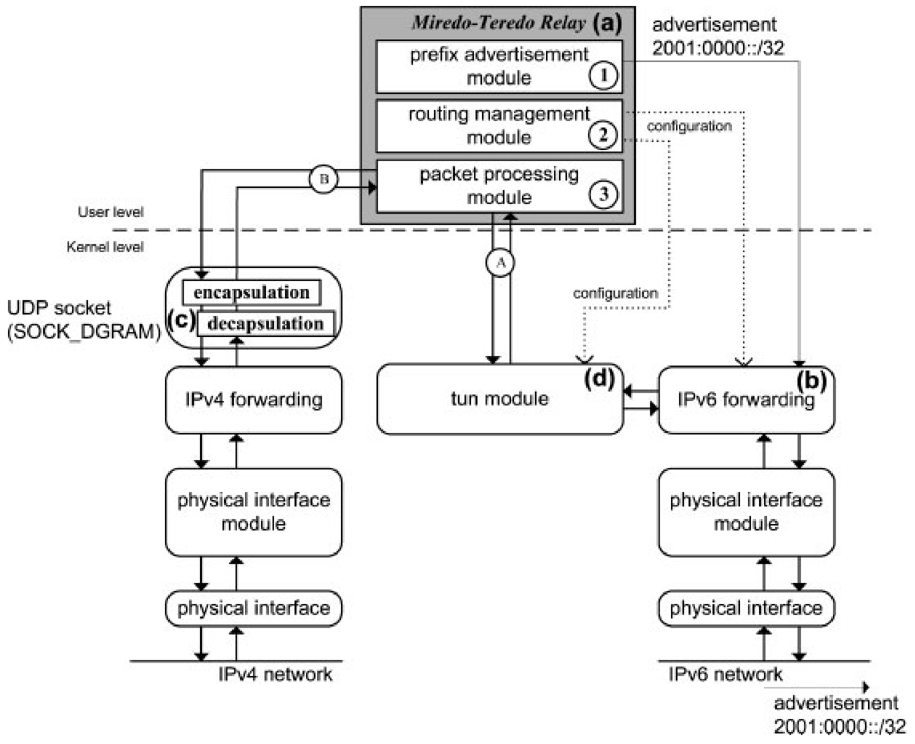


Fig. 12. Software architecture of the Miredo-Teredo relay.

The software architecture of the Miredo-Teredo relay (Figure 12(a)) is quite different from the previous two implementations. Specifically, the Miredo-Teredo relay relies on the IPv6 forwarding mechanism (Figure 12(b)), an IPv4 UDP socket (Figure 12(c)) and a *tun module* (Figure 12(d)) to provide IPv6 packet relay function. Unlike the NICI-Teredo relay that uses a kernel-level packet relay module, the Miredo-Teredo relay uses a user-level packet-processing module to send and receive the encapsulated packets. This Teredo relay consists of three components. The *prefix advertisement module* (Figure 12 ①) advertises the IPv6 address prefix to the IPv6 network, and the *routing management module* (Figure 12 ②) initializes the IPv6 forwarding plan to route the IPv6 packets to the Teredo clients through the *tun module*. The functions of these two modules are similar to those of NICI-Teredo relay and 6WIND-Teredo relay. The *packet-processing module* (Figure 12 ③) creates and maintains an IPv4 UDP socket for packet delivery with the Teredo clients, and utilizes the *tun module* for connection with the IPv6 protocol stack. This module delivers an IPv6 packet as the encapsulated datagram through the IPv4 UDP socket which binds to the port 3544. The *packet-processing module* provides IPv6 packet relay function by using `memcpy()` and socket functions (`sendto()` and `recvfrom()`) for IPv6 packets delivery between the *tun module* and the IPv4 UDP socket (see Figures 12 Ⓐ and Ⓑ). This architecture is clearly different from those of NICI-Teredo relay and 6WIND-Teredo relay.

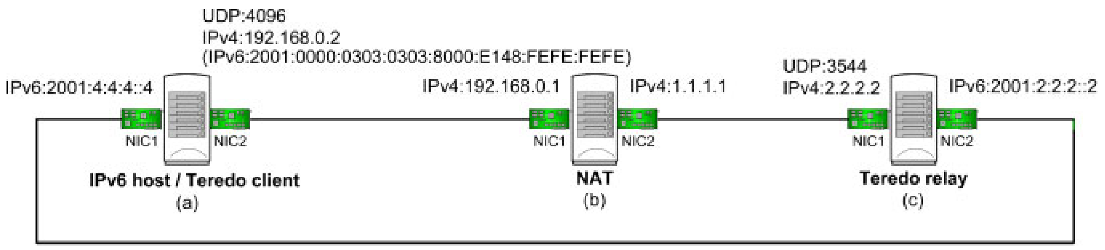


Fig. 13. The test environment.

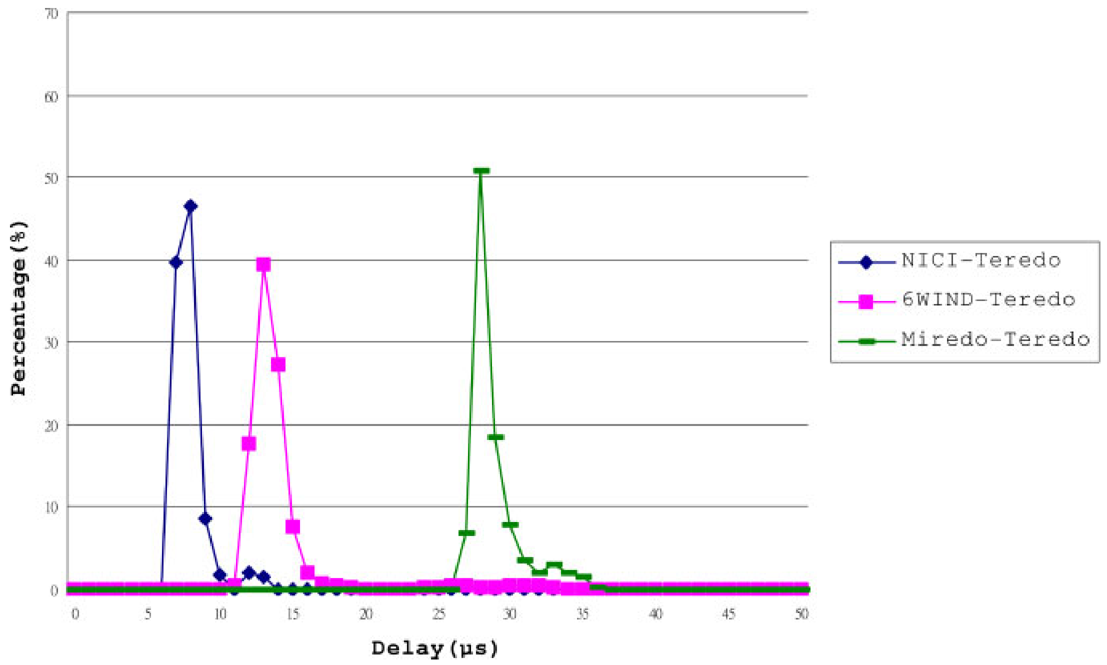


Fig. 14. IPv6 to IPv4 latency histograms of the Teredo relays (1280 bytes).

The Teredo relay design significantly affects the packet transmission performance. The packet processing latency of the NICI-Teredo relay is shorter because there are no packet copying operations between the kernel and the user levels. The performance comparison of these Teredo implementations will be elaborated in the next section.

As a remark, the NICI-Teredo relay is implemented as a loadable kernel module [21]. Although kernel hacking effort is required in developing the NICI-Teredo relay, the installation process is very simple and the users do not need to modify or re-compile the Linux kernel.

5. Performance Evaluation of Teredo Relay Implementations

Teredo relay handles large volume of network traffic and therefore is likely to be the bottleneck component in the Teredo mechanism. In this section, we investigate the performance of the three Teredo relay implementations described in the previous section. The output measurement is *packet-processing latency*, that is, the processing time at the Teredo relay, which is either the IPv6 to IPv4 latency (from a public IPv6 host to a private Teredo client) or the IPv4 to IPv6 latency (from a private Teredo client to a public IPv6 host).

The measurement environment consists of three hosts as illustrated in Figure 13. This environment follows the testing architecture in RFC 2544 [22] where a tester (Figure 13(a)) is configured as both the Teredo client function and the IPv6 host. This IPv6 host runs on Redhat Linux 9 with an IPv4 UDP daemon to simulate the Teredo client function. The NAT (Figure 13(b)) runs on Redhat Linux 9 with address mapping rules set by *iptables* [23]. The devices under test (DUTs) are the three Teredo relay implementations (Figure 13(c)): NICI-Teredo (version 0.5), 6WIND-Teredo (version 1.13) and Miredo-Teredo (version 0.4.1). Both NICI-

Table I. Average processing latency.

Teredo relay	Average IPv6 to IPv4 latency (µs)			Average IPv4 to IPv6 latency (µs)		
	64 bytes	512 bytes	1280 bytes	64 bytes	512 bytes	1280 bytes
NICI	7.56	7.67	7.91	8.47	8.71	8.93
6WIND	13.30	13.53	14.00	14.34	14.62	14.80
Miredo	28.55	29.45	30.26	27.94	28.76	29.69

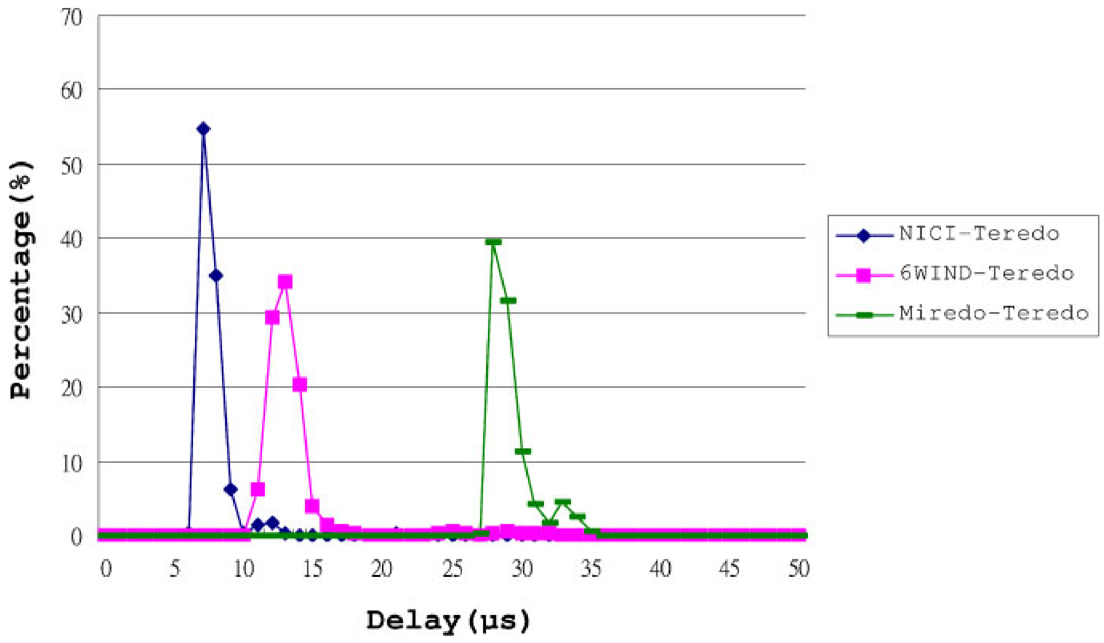


Fig. 15. IPv6 to IPv4 latency histograms of the Teredo relays (512 bytes).

Teredo relay and Miredo-Teredo relay run on Redhat Linux 9, while 6WIND-Teredo relay runs on FreeBSD 4.9. The hardware for the Teredo relay in Figure 13 is a personal computer with 1800+ AMD Athlon CPU, 256 MB SDRAM and two RealTek 8139 100BaseTx Ethernet cards.

In our measurement, a C program invoking *pcap* library [24] is used for catching the packet receiving and sending timestamps. We send one packet per second to the Teredo relay from IPv6 to IPv4 or IPv4 to IPv6, and measure the packet processing latency. Tests are conducted with three different packet sizes

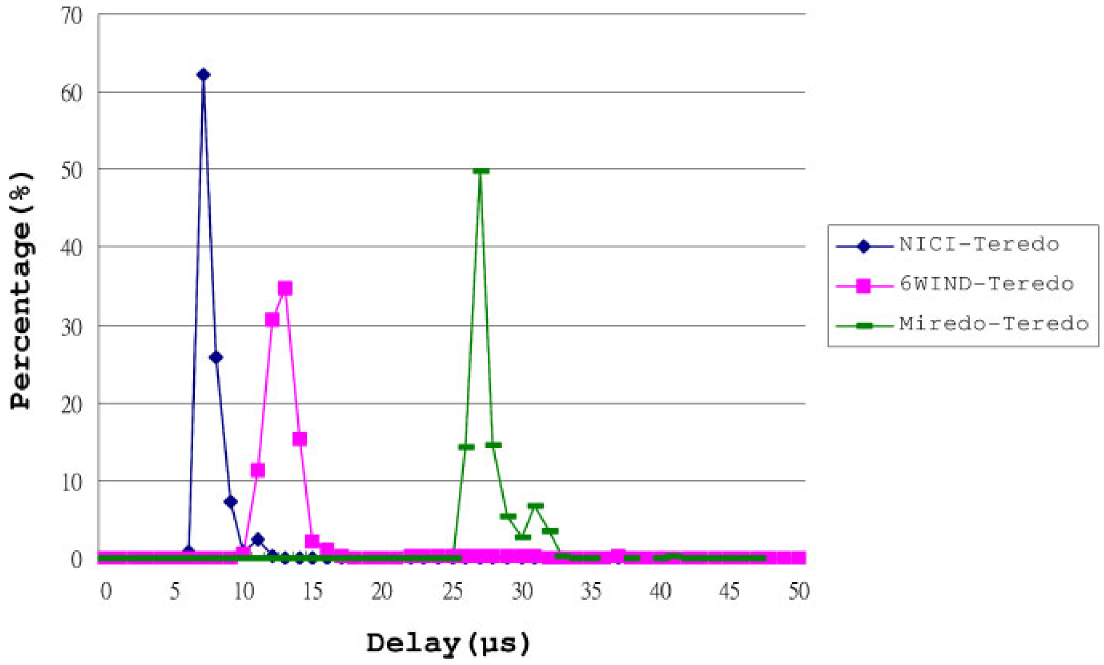


Fig. 16. IPv6 to IPv4 latency histograms of the Teredo relays (64 bytes).

(64, 512, and 1280 bytes, where the 1280-byte packet is the recommended IPv6 MTU size for Teredo [13]). Each test generates 10 000 packets to measure the IPv6 to IPv4 and IPv4 to IPv6 latencies.

Figure 14 shows the 1280-byte IPv6 to IPv4 latency histograms of the three Teredo relays. In this figure, the *x*-axis represents the delay time (in μs) and the *y*-axis represents the percentage of the packets (out of the 10 000 testing packets) that have this delay value. The latency of the NICI-Teredo relay is clustered around 7–9 μs and 12–13 μs. The latency of the 6WIND-Teredo relay is around 12–16 μs. The latency of the Miredo-Teredo relay is clustered around 27–31 μs and 33–35 μs. The 1280-byte IPv4 to IPv6 latency histograms are similar to those in Figure 14, and the average packet processing latency of the three Teredo relays are listed in Table I.

With different packet sizes (1280, 512, 64 bytes), the IPv6 to IPv4 latency histograms of NICI-Teredo relay and 6WIND-Teredo relay (see Figures 14–16) have similar shapes, where the Miredo-Teredo relay has longer latency than that of the NICI-Teredo relay. For the 1280-byte IPv6 to IPv4 latency listed in Table I, the latency improvement of the NICI-Teredo relay over the 6WIND-Teredo relay is around 44%, and the improvement of the NICI-Teredo relay over the Miredo-Teredo relay is around 74%.

6. Conclusion and Future work

This paper addressed the issues when dual-stack UEs hand off or roam between the IPv6 UMTS and the private IPv4 GPRS networks. Especially, we addressed the NAT traversal issue between the GPRS (private IPv4) and the UMTS (IPv6) networks. To solve this problem, we combined the SIP mobility mechanism and an automatic IPv6 tunneling mechanism, called Teredo, to support the handoff of a UE between different networks. As an NAT traversable automatic tunneling mechanism, Teredo provides convenient IPv6 access from the private IPv4 networks. We developed an efficient implementation for Teredo tunneling called NICI-Teredo. To our knowledge, NICI-Teredo was the first non-commercial Teredo implementation on Linux. Our approach has shorter packet processing latency than that of the 6WIND-Teredo relay (FreeBSD-based) and the Miredo-Teredo relay (Linux-based). The advantage of packet processing performance of the NICI-Teredo relay makes it an appropriate IPv6 tunneling solution.

As a final remark, we point out that Teredo does not work for all NAT servers. According to the rules for port mapping and access control in NAT [17], the four major types are full cone NAT, restricted cone NAT, port restricted cone NAT, and symmetric NAT. Although Teredo can successfully traverse the first three types of NATs, it fails in traversing symmetric NAT. To enhance Teredo for symmetric NAT is still an open issue for further study.

References

1. Rosenberg J, Schulzrinne H, Camarillo G, Johnston A, Peterson J, Sparks R, Handley M, Schooler E. SIP: Session Initiation Protocol. RFC 3261, June 2002.
2. 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Stage 2 (Release 5). Technical Specification 3GPP TS 23.228 V5.13.0 (2004–12), December 2004.
3. 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; General Packet Radio Service (GPRS); Service description; Stage 2 (Release 5). Technical Specification, 3GPP TS 23.060 V5.10.0 (2005–03), March 2005.
4. Lin Y-B, Huang Y-R, Pang A-C, Chlamtac I. ALL-IP approach for UMTS third-generation mobile networks. *IEEE Network Magazine* 2002; 16(5): 8–19.
5. Lin Y-B, Chlamtac I. *Wireless and Mobile Network Architectures*. John Wiley & Sons: New York, 2001.
6. Chen Y-K, Lin Y-B. IP connectivity for gateway GPRS support Node. *IEEE Wireless Communications Magazine* 2005; 12(1): 37–46.
7. Gilligan R, Nordmark E. Transition Mechanisms for IPv6 Hosts and Routers. RFC 2893, August 2000.
8. Thakolsri S, Prehofer C, Kellerer W. Transition Mechanism in IP-based Wireless Networks. In *Proceedings of 2004 IEEE International Symposium on Applications and the Internet Workshops* (SAINT Workshops 2004), 2004; pp. 112–119.
9. Carpenter B, Moore K. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056, February 2001.
10. Durand A, Fasano P, Guardini I, Lento D. IPv6 Tunnel Broker. RFC 3053, January 2001.
11. Srisuresh P, Holdrege M. IP Network Address Translator (NAT) Terminology and Considerations. RFC2663, August 1999.
12. Levkowetz H, Vaarala S. Mobile IP Traversal of Network Address Translation (NAT) Devices. RFC 3519, April 2003.
13. Huitema C. Teredo: Tunneling IPv6 over UDP through NATs. RFC 4380, February 2006.
14. Liu M, Wu X, Cai Y, Jin M, Li D. Tunneling IPv6 with private IPv4 addresses through NAT devices. draft-liumin-v6ops-silkroad-03.txt (expired), July 2005.
15. Schulzrinne H, Wedlund E. Application-layer mobility using SIP. *ACM SIGMOBILE Mobile Computing and Communications Review* 2000; 4(3): 47–57.
16. Banerjee N, Wu W, Das SK, Dawkins S, Pathak J. Mobility support in wireless Internet. *IEEE Wireless Communications Magazine* 2003; 10(5): 54–61.
17. Rosenberg J, Weinberger J, Huitema C, Mahy R. STUN—Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, March 2003.
18. Huang S-M, Wu Q-C. Implementation of Teredo—Tunneling IPv6 through NAT. *Technical Report for National Information and Communication Initiative (NICI) IPv6 R&D Division*, Taiwan, ROC, 2003.
19. Teredo for FreeBSD. http://www-rp.lip6.fr/teredo/

20. Miredo: Teredo for Linux. http://www.simphalempin.com/dev/miredo/
21. Henderson B. Linux Loadable Kernel Module HOWTO. http://www.linux.org/docs/ldp/howto/module-howto/
22. Bradner S, McQuaid J. Benchmarking Methodology for Network Interconnect Devices. RFC 2544, March 1999.
23. The netfilter/iptables project. http://www.netfilter.org/
24. Tcpdump and libpcap. http://www.tcpdump.org/

Authors' Biographies

Shiang-Ming Huang received his B.S. and M.S. degrees from National Chiao Tung University, Hsinchu, Taiwan, in 2003 and 2005 respectively. He wrote the STUN protocol dissector in Ethereal, which is a popular open source network protocol analyzer distributed all over the world. In 2003, he helped Computer and Communication Research Laboratories (CCL) to enhance a commercial SIP User Agent to support STUN, which is an important mechanism proposed by Internet Engineering Task Force (IETF) in RFC 3489. This enhancement was then transferred to Industrial Technology Research Institute (ITRI) later in the same year. After that, he joined the National Information and Communications Initiative Committee (NICI) IPv6 deployment project in 2003, where he implemented the first non-commercial Teredo IPv6 tunneling mechanism on Linux. Currently, he is working towards his Ph.D. in Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His current research interests include Internet Protocol version 6 (IPv6) and Session Initiation Protocol (SIP).

Quincy Wu received his B.S. degree in Mathematics from National Tsing Hua University in 1992, and his Ph.D. in Computer Science and Information Engineering from National Tsing Hua University in 2000. He joined National Center for High-Performance Computing with the NBEN (National Broadband Experimental Network) project, where he successfully designed and established the first island-wide IPv6 network among universities. In 2002, he began serving as a research assistant professor with National Chiao Tung University, and helped National Telecommunications Project Office to deploy a SIP-based VoIP Platform across several universities. Since 2004, he co-chairs the SIP-H323 Working Group of Asia-Pacific Advanced Network (APAN) and helped Taiwan Academic Network (TANet) to design and deploy VoIP services. He was appointed as an assistant professor of Graduate Institute of Communication Engineering, National Chi Nan University in 2005 and helped initiating the VoIP over WiMAX project in National Chi Nan University. His current research interests include session initiation protocol, open service architecture, Internet protocol version 6, design and analysis of approximation algorithms, and service creation on the third generation mobile network.

Yi-Bing Lin is chair professor and vice president of Research and Development, National Chiao Tung University. His current research interests include wireless communications and mobile computing. Dr. Lin has published over 190 journal articles and more than 200 conference papers. Lin is the co-author of the book *Wireless and Mobile Network Architecture* (with Imrich Chlamtac; published by John Wiley Sons). Lin is an IEEE fellow, an ACM fellow, an AAAS fellow, and an IEE fellow.

Che-Hua Yeh received his B.S. degree from National Chiao Tung University, Hsinchu, Taiwan in 2004. He had rich research experience on SIP instant messaging and GSM short message service. Since 2003, he joined the 'Personal Communication System' research group in National Chiao Tung University and worked on the project 'Instant Messaging and Short Messaging Service Gateway' (IM-SMS Gateway) which implemented a gateway converting instant messaging service and short messaging service between IP network and GSM network. After that, he developed the Intelligent Notify Center (iNotify Center) which integrates SIP instant messaging, SIP presence service, GSM short messaging service, and E-mail service. He currently studies the subject of Session Initiation Protocol (SIP) mobility and implements SIP mobility in SIP user agent. He is now working towards the Master degree in Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan.