NCTU SLT: A Socket-layer Translator for IPv4-IPv6 Translation

Whai-En Chen, *Member, IEEE*, Chia-Yung Su and Yi-Bing Lin, *Fellow, IEEE*

Dept. of Comp. Sci. and Info. Engr., National Chiao Tung Unviersity
{wechen,cysu,liny}@csie.nctu.edu.tw

Abstract

Based on *Bump-In-the-API* (BIA) architecture, this letter proposes a Socket-layer Translator called NCTU SLT that translates IPv4 applications to IPv4/IPv6-capable applications on dual-stack hosts. Like previously proposed approaches, NCTU SLT can translate NAT-friendly applications (e.g., HTTP and TELNET) without modifying the source codes of the IPv4 applications. Furthermore, NCTU SLT has the advantages over the previous approaches for translating non-NAT friendly applications (i.e., SIP and FTP).

Keywords: ALG, BIA, BIS, IPv6 Transition, Socket-layer Translator

I. Introduction

In the next generation network, IPv6 will be widely deployed to offer large address space and support new features such as security, mobility, QoS, and plug-and-play. To maintain compatibility with IPv4 while deploying IPv6, transition mechanisms such as *dual-stack* have been proposed. A dual-stack host installed both IPv4 and IPv6 protocol stacks can choose either IPv4 or IPv6 to communicate with an IPv4 or an IPv6 host. In the early IPv6 deployment stage, many backbone networks only support IPv4, and IPv6 hosts are "islands" in these IPv4 networks. In such environments, *translation* technique provides interoperability between IPv4-only and IPv6-only applications. In this letter, we propose a translation approach called *Socket-layer Translator* to translate IPv4-only applications to IPv6 on a dual-stack device.

Microsoft Windows XP and 2003 are dual-stack operating systems. They also provide *Windows Socket* (Winsock) *Application Programmable Interfaces* (APIs) for developing network applications. However, some Winsock APIs and parameters of IPv4 are different from that of IPv6. As a result, IPv4 applications can only use IPv4 stack even if the operating system is upgraded to dual-stack. To support IPv6 or IPv4/IPv6 dual modes, programmers must manually check and modify the source codes. This translation task is tedious.

To smoothly translate IPv4 applications, IETF RFC specifies *Bump-In-the-Stack* (BIS) [1] and *Bump-In-the-API* (BIA) [2] that allow the dual-stack hosts to use IPv4 applications to access IPv6 resources. Both mechanisms can successfully translate NAT friendly protocols (e.g. HTTP and TELNET) [4]. Some protocols such as *File Transport Protocol* (FTP) and *Session Initiation Protocol* (SIP) contain IP address and port information in the application-layer headers. Neither BIS nor BIA process the application-layer headers, and therefore they cannot translate IPv4 FTP and SIP applications into IPv6.

To support IPv6 transition for FTP and SIP applications, we develop a Socket-layer Translator called *NCTU* Socket-layer Translator (*SLT*). Based on a BIA-like architecture, NCTU SLT translates NAT friendly protocols by redirecting IPv4 socket functions to IPv6. For non-NAT friendly protocol that cannot be translated by BIS and BIA, we implement *Application Layer Gateways* (ALGs) and ALG Manager that allow the dual-stack hosts to access IPv6 resources through IPv4 applications without any modification.

II. System Architecture of Socket-layer Translator

Figure 1 shows the NCTU SLT and its implementation environment.

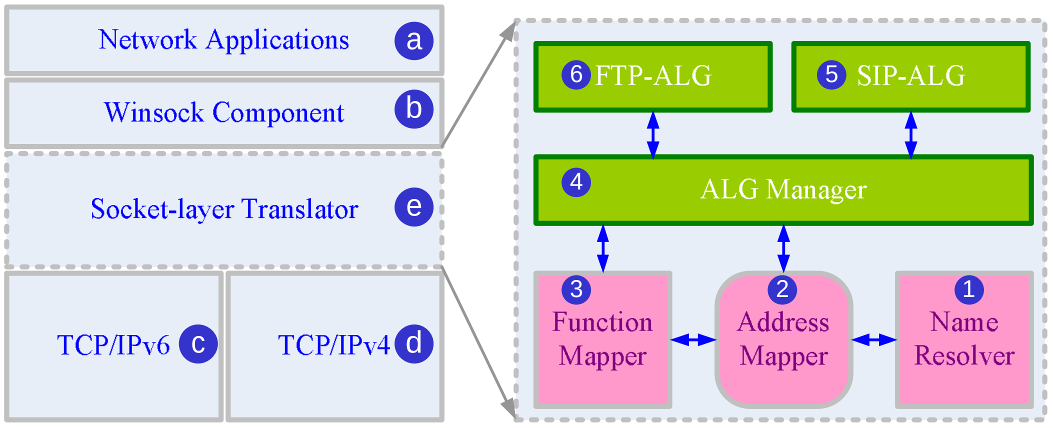

Figure 1. NCTU Socket-layer Translator and its Implementation Environment

Among the network applications supported by NCTU SLT (Figure 1ⓐ), HTTP and TELNET are NAT friendly protocols. On the other hand, FTP and SIP contain IP address and port information in the application-layer headers and therefore require modifications to their application-layer headers for IPv6 translation.

Microsoft Winsock Component (Figure 1ⓑ) provides APIs to develop network applications. IPv6 and IPv4 protocol stacks (Figure 1ⓒ and ⓓ) are implemented as *Transport Service Providers* (TSPs) in the Windows systems, and each of the IPv4/IPv6 transport protocols is implemented as a *Layered Serviced Provider* (LSP). LSPs are linked as a protocol chain in TSP. The NCTU SLT (Figure 1ⓔ) is implemented as a LSP embedded in TCP(UDP)/IPv4 TSP. Therefore the NCTU SLT can intercept the functions invoked by IPv4 applications and translated them into equivalent IPv6 functions.

The NCTU SLT consists of six components. The *Name Resolver* (Figure 1①) intercepts the *Domain*

Name System (DNS) functions (e.g., gethostbyname() and gethostbyaddr()) from IPv4 applications and redirects them to IPv6/IPv4 DNS functions (e.g., getaddrinfo() and getnameinfo()).

The trick that allows an IPv4 application to access IPv6 resources is played in the *Address Mapper* (Figure 1②). The Address Mapper contains an IPv4-IPv6 mapping table and a pool of *exclusive private IPv4* (EPI) *addresses* (e.g., 10.0.0.1~10.0.0.255). We use FTP/SIP examples to illustrate how NCTU SLT works.

Example 1: When an IPv4 application with NCTU SLT accesses an IPv6 FTP server through the server's domain name, the Name Resolver will obtain an IPv6 destination address through the IPv6 DNS queries mentioned before. This destination address cannot be recognized by the IPv4 application. Therefore the received IPv6 address is mapped to an IPv4 address from the EPI address pool, and this mapping is stored in the IPv4-IPv6 mapping table. The mapped EPI address is then returned to the IPv4 application. When the IPv4 application issues an IPv4 function (e.g., connect()) with the EPI address, the *Function Mapper* (Figure 1③) intercepts the function and translate the IPv4 parameters into IPv6, and instructs the Address Mapper to use the EPI address to retrieve the corresponding IPv6 address in the IPv4-IPv6 mapping table. The IPv6 address, which is a parameter of connect(), indicates the FTP destination. (On the other hand, if the IPv4 application accesses a non-EPI IPv4 address, the Address Mapper will not found the entry in the IPv4-IPv6 mapping table, and the original IPv4 address is used to access the IPv4 destination.) To send the packets to the FTP server, the IPv4 application invokes the send() function including the data to be transmitted. The data may contain IP address/port information that needs IPv4-to-IPv6 translation by FTP-ALG to be elaborated in **Example 3**.

Example 2: To allow IPv6 clients to access an IPv4 FTP server with NCTU SLT, the FTP server invokes IPv4 Winsock function bind() to associate the server's IPv4 address with an IPv4 socket. On this socket, the recv() function is activated to receive the incoming IPv4 packets. When the IPv4 bind() is invoked, the Function Mapper intercepts the parameters (e.g., AF_INET and INADDR_ANY) and translates them to IPv6 (AF_INET6 and in6addr_any). Then the IPv6 bind() is invoked to bind the server's IPv6 address to an IPv6 socket. The Function Mapper also intercepts the IPv4 recv() and then invokes IPv6 recv() to obtain data on IPv6 socket. At this point, the FTP server can also receive the incoming IPv6 packets. FTP-ALG translation for data receiving is similar to that for data sending in **Example 1**.

The major contributions of this letter are the design and implementation of ALG-related mechanisms in NCTU SLT. The *ALG Manager* (Figure 1④) checks the application-layer messages intercepted by the Function Mapper, and dispatches these messages to a proper ALG according to the transport port number. For example, the port number for FTP-ALG is 21 and the port number for SIP-ALG is 5060.

Example 3. The *FTP-ALG* (Figure 1⑥) translates both IP/port information in the FTP messages and the FTP commends. The IPv4 command PORT is translated to IPv6 EPRT, PASV is translated to EPSV, and the response 227 is translated to 229. The IPv4 address (A1,A2,A3,A4) in PORT command is translated to "net-addr" (an IPv6 address) in the EPRT command. Also, the port numbers (p1,p2) in the PORT command and 227 response are translated to another format ("tcp-port"= p1*2^8+p2) in the EPRT command and 229 response.

Example 4. *SIP-ALG* (Figure 1⑤) translates the IP address information in the SIP and *Session Description Protocol* (SDP) header fields. A SIP *User Agent* (UA) utilizes SIP to transmit signaling information and *Real-time Transport Protocol* (RTP) to transmit voice packets. An IPv4 SIP UA with NCTU SLT can communicate with an IPv6 UA through SIP-ALG translation. When the IPv4 SIP UA sends out a SIP message, the SIP-ALG intercepts this message, and translate IP addresses in SIP Request URI, To, Contact, Record-Route, and Route header fields. Also, in the SIP message body, SDP c header field contains IP address information of the RTP connections. All IPv4 addresses contained in the header fields mentioned above are EPI addresses. (The IPv6-EPI address mappings for these addresses are established through a procedure similar to that in **Example 1**.) The SIP-ALG invokes the Address Mapper to map the EPI addresses to the corresponding IPv6 addresses. These IPv6 addresses are appropriately included in the SIP and the SDP header fields before the SIP message is sent out.

III. Performance Evaluation

This section compares four IPv4-IPv6 translation mechanisms. In *manual modification*, a programmer upgrades IPv4 applications to IPv6 by directly modifying the source codes. The programmer can utilize Microsoft checkv4.exe to search the IPv4-related functions, data structures and parameters. However, user-defined functions, structures, and variables cannot be detected by this program. Therefore, the programmer should trace whole source codes in most cases.

BIS, BIA and NCTU SLT are designed to translate IPv4 applications without modifying the source codes. Inserted between the TCP/IP protocol stack and network card drivers. BIS intercepts IPv4 packets and then translates these packets into IPv6 format according to the SIIT algorithm [3]. Note that BIS cannot support peer-to-peer security. BIA and NCTU SLT are inserted between the socket component and the TCP/IP protocol stack. Therefore they can translate IPv4 socket functions to IPv6 without modifying the packet format, and IPsec for peer-to-peer security can be supported.

We conduct experiments to measure the NCTU SLT performance on FTP applications transfer. In our experiments, the FTP server is vsftpd running Linux Fedora Core 1. The FTP client is SmartFTP running on Windows XP SP1. We consider three scenarios. **Scenario 1** downloads pure IPv4 packets from the server to the client. **Scenario 2** downloads pure IPv6 packets from the server to the client. **Scenario 3** downloads packets from the IPv6 server to the IPv4 client through NCTU SLT. Files with sizes ranging from 10KB to 2GB are downloaded from the FTP server via the 100Mbps Ethernet. Figure 2 plots the transmission delays of the three scenarios against various file sizes. The figure indicates that performance degradation due to NCTU SLT translation is insignificant. For example, in downloading 2GB file, **Scenario 1** takes 205 seconds, while **Scenario 3** only requires 4 more seconds over **Scenario 2** and 15 seconds over **Scenario 1**. In other words, the NCTU SLT degradation is less than 2% over pure IPv6 transmission (i.e., code modification) and less than 8% over IPv4 transmission.

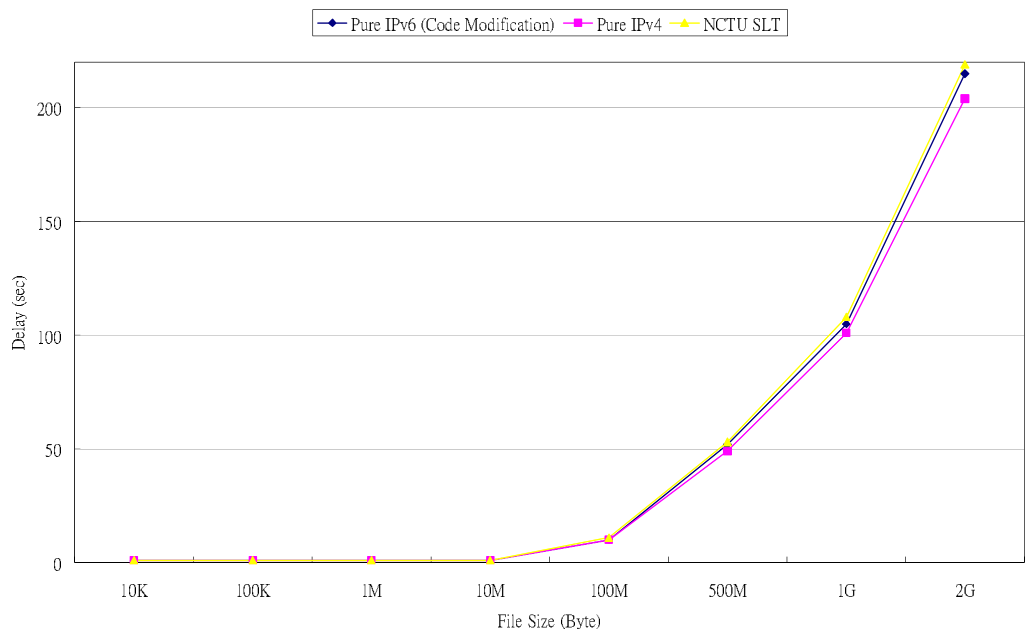

Figure 2. Transmission Delay with and without NCTU SLT

Acknowledgement This work was sponsored in part by Taiwan NICI IPv6 Steering Committee, R&D Division under contract number R-0400, NSC Excellence project

NSC93-2752-E-0090005-PAE and NTP VoIP Project under grant number NSC 92-2219-E-009-032.

Reference

[1] Dual Stack Hosts using the Bump-In-the-Stack Technique (BIS). K. Tsuchiya, H. Higuchi, Y. Atarashi. IETF RFC2767. February 2000.

[2] Dual Stack Hosts Using Bump-in-the-API (BIA). S. Lee, M-K. Shin, Y-J. Kim, E. Nordmark, A. Durand. IETF RFC3338. October 2002.

[3] Stateless IP/ICMP Translation Algorithm (SIIT). E. Nordmark. IETF RFC2765. February 2000.

[4] Network Address Translator (NAT)-Friendly Application Design Guidelines. D. Senie. IETF RFC3235. January 2002.