Kernel Module Implementation of IPv4/IPv6 Translation Mechanisms for IPv4-oriented applications

Katsuhiro Naito, Kazuo Mori, and Hideo Kobayashi
Department of Electrical and Electronic Engineering, Mie University,
1577 Kurimamachiya, Tsu, 514-8507, Japan
Email: {naito, kmori, koba}@elec.mie-u.ac.jp

***Abstract*—Only IPv6 addresses are currently being assigned to hosts because IPv4 addresses will be exhausted in the near future. However, almost all network applications still lack support for IPv6 communication. Therefore, users will suffer from the unavailability of IPv6 oriented applications. Bump-In-the-Stack (BIS) mechanisms can allow hosts to communicate with other hosts through IPv6 networks using existing IPv4-oriented applications. These mechanisms will be required to achieve a smooth transition from IPv4 to IPv6 networks in the near future. However, detailed implementation schemes are dependent upon the operating system. Additionally, since conventional network address translation mechanisms usually perform in a user space, throughput performance degrades as a result of the memory copy between kernel space and user space. Recently, Session Initiation Protocol (SIP) has been used to achieve multimedia communication. However, BIS does not support address translation mechanisms for embedded IP addresses in packet payload, such as in SIP messages. This paper presents a specially developed Linux kernel module for IPv4/IPv6 address translation supporting SIP messages. The kernel module can hook all packets in a Linux network socket using Linux netfilter mechanisms. The advantages are high throughput, as the memory copy is limited to a socket buffer in a Linux network stack, and flexible installation to an original Linux kernel. Thus, the kernel module allows users to achieve IPv4/IPv6 address translation by installing it in a generic Linux kernel, without modifying the kernel source.**

***Keywords*-Bump-In-the-Stack; Session Initiation Protocol; Kernel module; Address translation; Linux.**

I. Introduction

The Internet will soon exhaust another IPv4 address range. Recently, the Asia-Pacific Network Information Center (APNIC) announced that the APNIC pool had reached its final /8 IPv4 address block [1]. Hence, only the IPv6 address range will be assigned to new networks in the near future.

IPv6 is the network layer protocol for the next generation Internet and offers a much larger address space. Since networking equipment vendors have developed IPv6 implementation, almost all networking equipment for enterprise networks already supports IPv6 communication. However, IPv6 is a different protocol from IPv4. Furthermore, there is still a great deal of IPv4 content on the Internet. While IPv4 devices and services continue to be widespread, it is difficult to replace IPv4 with IPv6. Therefore, we are entering a transition period during which network address translation (NAT) mechanisms will be required to communicate between IPv4 and IPv6 networks [2], [3].

Various translation mechanisms for IPv4/IPv6 have been proposed to facilitate the interoperation and coexistence of both protocols [4]. Dual-stack lite requires an IPv6 access network and tunnels between a host and a Network Address Port Translation (NAPT) 44 device, which is operated by service providers [5]. The dual-stack host, which has both an IPv4 and an IPv6 address, sends its IPv4 traffic through a NAPT44 device even though the service provider's access network is IPv6. Additionally, the host can send its IPv6 traffic routed normally. In dual-stack lite, hosts require an IPv4 address and an IPv6 address. Therefore, it will be difficult to apply in the near future because new IPv4 addresses will have been exhausted.

The other candidate mechanisms are NAT64 and NAT-PT [6], [7], [8], which are translation mechanisms where the host runs only IPv6. They are called large-scale NATs (LSNs) or carrier grade NATs. Recent NAT64 devices can serve a translation function to 10,000 subscribers, but their scalability will be limited due to the expansion of network traffic. In addition, the translations have several technical issues [9].

These mechanisms can translate between IPv4 and IPv6 packets. However, applications also need to support IPv6 addresses in order to use IPv6 networks. Moreover, since modification of the source code is required before IPv4-oriented applications can support IPv6 addresses, almost all these applications still cannot do so.

Bump-In-the-Stack (BIS) allows hosts to communicate with other IPv6 hosts using existing IPv4 applications [10]. However, the BIS implementation is unable to provide high throughput and flexible installation, while application protocols that embed IP addresses in the packet payload are not supported. Since Session Initiation Protocol (SIP) messages include host's IP addresses, translators need to modify the IP address part in such messages if they are to support SIP [11], [12]. Session Traversal Utilities for NAT (STUN) [13] and Universal Plug and Play (UPnP) [14] are well known tools in the context of a NAT traversal solution. However, they are difficult to apply for translation between IPv4 and IPv6 addresses because we have to assume that SIP client applications do not support IPv6 addresses. Therefore, the SIP Application Level Gateway (ALG) is a better solution for translation between IPv4 and IPv6 addresses.

In this paper, we develop a kernel module for Linux

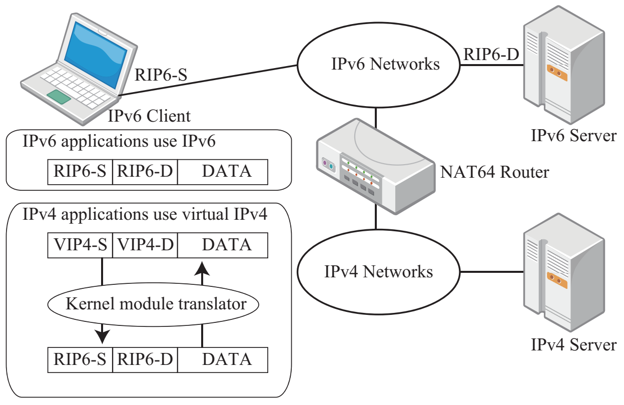

Figure 1. Overview network.

netfilter [15]. The developed kernel module can translate between IPv4 and IPv6 addresses in a header and modify addresses in SIP messages. Since the developed kernel module performs packet manipulation in a kernel space, we can achieve high throughput performance, even with IPv4/IPv6 address translation, by reducing the memory copy of packet data. Furthermore, the developed kernel module can be implemented in Linux OS without kernel modification.

II. IPv6/IPv4 Translation Mechanisms

Figure 1 shows an overview of the network presented in this paper. Here, we focus on hosts with IPv6 addresses, which represent the reality in the near future, as discussed above. As it is difficult for users to modify the source code of applications to support IPv6 networks, these hosts also have IPv4 oriented applications. However, IPv4 oriented applications cannot establish connections because the hosts do not have IPv4 addresses.

The proposed implementation provides two virtual IPv4 addresses: a source IPv4 address for a virtual network interface and a destination IPv4 address corresponding to a destination host for IPv4 applications. Thus, IPv4 applications can establish connections using virtual IPv4 addresses. During real communication, these virtual IPv4 addresses are translated to corresponding real IPv6 addresses. As a result, IPv6 applications can communicate using real IPv6 addresses, while IPv4-oriented application can communicate using virtual IPv4 addresses.

The fundamentals of IPv6/IPv4 translation mechanisms are discussed in BIS. But the implementation method is not described because it is specific to the operating system. Additionally, SIP is usually used for multimedia communications, such as voice or video conference applications. However, as these SIP applications depend on service providers, it is difficult for the user to select optimum SIP applications that will support IPv6 communication. In this paper, we extend the BIS mechanisms to support SIP applications, clarify the design for implementation, and develop a special kernel module for Linux OS.

Figure 2 shows the system model for packet manipulation in the developed kernel module. The functions of this module are classified into address translation function, payload modification function, and DNS message handling function. The kernel module uses the Linux netfilter function to handle a socket buffer for each packet. Therefore, modification of the original Linux kernel is not required in order to use the developed kernel module.

A. *Virtual Interface*

In the developed kernel module, instead of a real IPv6 address, IPv4-oriented applications use a virtual IPv4 address that is allocated in the network interface. Therefore, some network interface for the virtual IPv4 address is required to transmit packets with the virtual IPv4 address as a source address.

In the proposed implementation, we create a virtual network interface for the virtual IPv4 address using tun/tap interfaces. Tun is software emulation of ethernet devices and tap is software emulation of a network layer. Usually, tap is used for creating a bridge interface and tun is used for creating tunnels. However, since in our proposed implementation the virtual interface is used to assign the virtual IPv4 address, both mechanisms are available. Additionally, the developed kernel module can hook all packets from IPv4 oriented applications. Therefore, the virtual interface does not receive any packets from IPv4 oriented applications.

B. *Packet hook in Linux netfilter*

Netfilter provides a packet manipulation framework inside the Linux 2.4.x and 2.6.x kernel series, and it is also a set of hooks inside the Linux kernel. Therefore, kernel modules can register their callback function with the Linux network stack and the function is called when packets traverse the respective hook points. As netfilter also allows kernel modules to send the hooked packets back to the network stack, these modules can modify packet information without modification of the original Linux kernel.

In the developed kernel module, outbound packets from both IPv4 and IPv6 applications are hooked at the point NF_INET_LOCAL_OUT. In the Linux network stack, IPv4 and IPv6 are processed separately. Therefore, the developed kernel module receives both IPv4 and IPv6 packets separately from the point NF_INET_LOCAL_OUT. Whereas IPv6 packets from IPv6 applications are sent back to the Linux network stack immediately, at the point NF_INET_POST_ROUTING, IPv4 packets from IPv4-oriented applications undergo some manipulation, in respect of address translation and payload modification, before being sent back to the latter point. A similar differentiation is made for inbound packets, where the respective stack points are NF_INET_POST_ROUTING and NF_INET_LOCAL_IN. Thus, IPv6 applications engage in real IPv6 communication in the normal way, while IPv4 oriented applications perform virtual IPv4 communication through IPv6 networks.

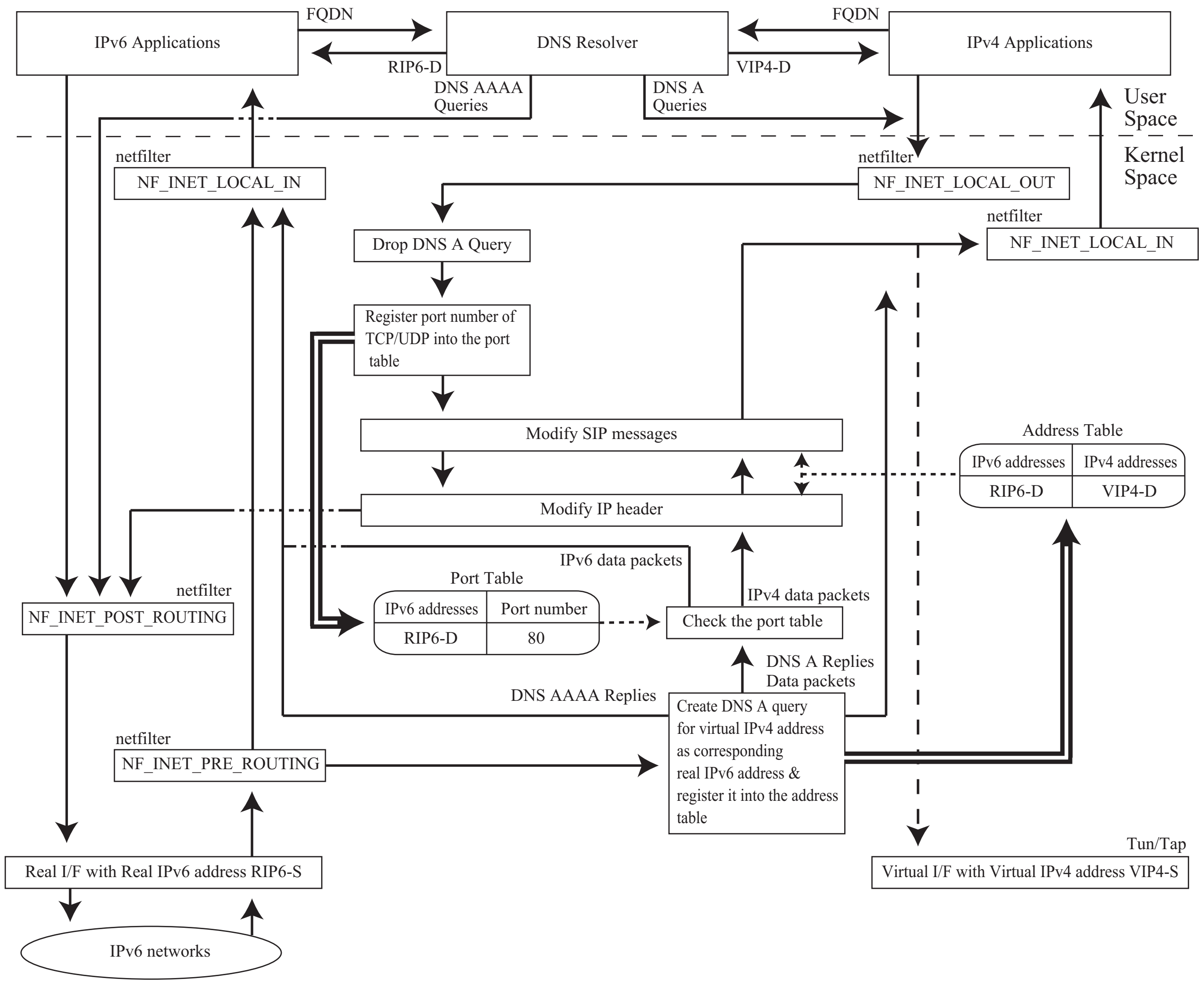

Figure 2. Design of packet manipulation in kernel module.

III. Assignment of virtual IPv4 address

In the proposed implementation, a source IPv4 address for IPv4-oriented applications is allocated as a predefined virtual IPv4 address, $VIP4-S$, and a source IPv6 address for physical network interface is allocated as a predefined real IPv6 address, $RIP6-S$. Additionally, a destination IPv4 address corresponding to a real IPv6 address is assigned dynamically when a DNS reply message is received. These IPv4 addresses consist of private addresses and are used internally in the operating system; thus, the address assignments do not negatively affect other hosts. This subsection describes the procedure for virtual IPv4 assignment.

A. *Translation of DNS messages*

- Resolution of Fully Qualified Domain Name (FQDN)
  When IPv4 oriented applications try to communicate with a server host, the DNS resolver transmits a DNS AAAA query and a DNS A query to find an IP address corresponding to the FQDN.
- Discard of DNS A query
  The DNS A query is meaningless, because the host does not have a real IPv4 address. Therefore, the transmitted DNS A query is dropped in the kernel module.
- Creation of a virtual IPv4 address
  A new virtual IPv4 address corresponding to the real IPv6 address is required as a destination IP address for the IPv4-oriented application. Therefore, the DNS AAAA reply corresponding to the transmitted DNS AAAA query is hooked by the kernel module when it is received from the physical interface. The kernel module creates a new virtual IPv4 address, $VIP4-D$, corresponding to the real IPv6 address, $RIP6-D$, in the DNS
- Registration of the IPv4/v6 address pair
  Since information about the pair of virtual IPv4 address

and the real IPv6 address is required in order to modify the IP header, the kernel module registers the pair of them in the address table.

- Response of virtual IPv4 address
  The DNS resolver returns the virtual IPv4 address corresponding to the FQDN by creating a DNS A reply. As a result, the IPv4-oriented application can communicate with the IPv6 host using the source virtual IPv4 address, $VIP4 - S$, paired with the destination virtual IPv4 address, $VIP4 - D$, while the host can communicate with the IPv6 server host using the source real IPv6 address, $VIP6 - S$, paired with the destination real IPv6 address, $VIP6 - D$.

B. Translation of IPv4/IPv6 addresses

This paper assumes that both IPv4-oriented applications and IPv6 applications communicate via IPv6 networks. This subsection describes the process for translation of IPv4/IPv6 addresses in the developed kernel module.

- Registration of IPv4 applications
  The developed kernel module receives IPv6 packets for both IPv4-oriented applications and IPv6 applications. However, the IPv6 packets received do not have information corresponding to the IP version of the destination application. Accordingly, the kernel module employs a port table, where a destination IPv6 address and a port number are registered when the kernel module receives IPv4 packets from IPv4-oriented applications.
- Modification of transmitted packets
  The kernel module handles transmitted IPv4 packets as a socket buffer in the Linux network stack. Since the header size of IPv4 is different from that of IPv6, the kernel module extends the header space in the socket buffer and modifies the header information of IPv4 to conform to that of IPv6. As in the case of BIS mechanisms, the developed kernel module cannot be used with IPv4 applications that use any IPv4-specific option.
- Selection of received packets
  The kernel module checks the port table to determine whether the received packets are destined for IPv6 applications or IPv4 applications. Packets for IPv6 applications are sent directly back to the point NF_INET_LOCAL_IN. For IPv4 applications, however, before being sent back to that point, the packets undergo a process that reduces their header space in the socket buffer and modifies the IPv6 header information to conform with that of IPv4, according to the address table.

IV. Translation mechanisms for SIP

In usual network address translation, only IP addresses included in header information are modified. However, complete IP conversion also requires the translation of IP addresses embedded in application layer protocols, such as those found in File Transfer Protocol (FTP) and SIP. Since some implementations of FTP already support IPv6, FTP applications will be available in IPv6 networks. However, almost all SIP applications still do not support IPv6. Additionally, since the profiles of SIP applications depend on those of the service provider, it is difficult to modify SIP applications to suit a user's preferences.

The developed kernel module also supports translation mechanisms for SIP applications. The general application level 2 gateway for SIP applications only converts IP addresses within private networks, whereas the mechanisms proposed here need to convert IP addresses from virtual IPv4 addresses to real IPv6 addresses. Therefore, the conversion point in the packet payload is different from the usual cases.

Since messages from networks include real IPv6 addresses in the packet payload, the kernel module needs to convert real IPv6 addresses to virtual IPv4 addresses. In addition, messages from applications include virtual IPv4 addresses in the packet payload. Therefore, the kernel module also needs to convert virtual IPv4 addresses to real IPv6 addresses.

The following fields are converted in the kernel module.

- Via header
  A via header includes a client's host name or an IP address, and a port number at which it wishes to receive responses.
- Record-Route header
  A Record-Route header field includes a host name or an IP address of a proxy. It is usually used to log SIP traffic so as to charge a usage fee.
- Contact header
  A Contact header field value may contain a display name, a URI with URI parameters, and header parameters. It indicates a response host.
- Body
  A Session Description Protocol (SDP) field includes a client's host name or an IP address. It provides information about session identification and the types of data communication used in the session.

V. Numerical results

In order to evaluate the design of the developed kernel module, we measured throughput, standard deviation of throughput, and round trip time by changing the size of the maximum segment size (MSS) of the Transmission

Table I
Performance evaluation parameters.

OS	Linux
Distribution	Ubuntu 10.04
Kernel version	linux-2.6.32-24-generic
CPU	Intel Pentium 4 2.40GHz
Memory	512 MBytes
Application	iperf, nuttcp
Size of transferred data	200 MBytes
Transport protocol	TCP
Number of measurement	10

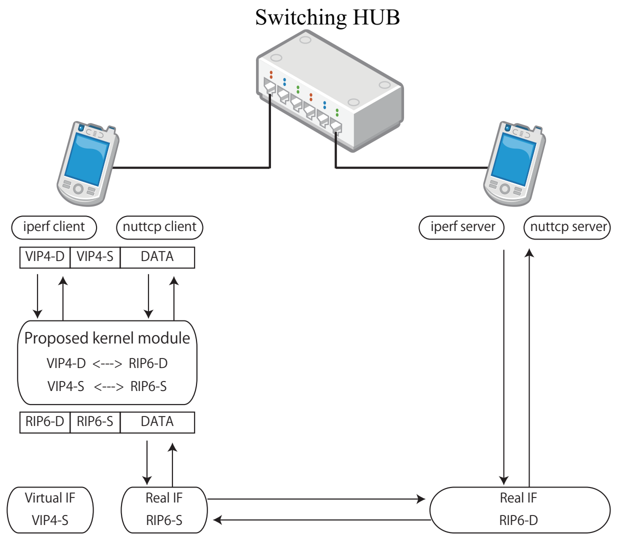

Figure 3. Evaluation model.

Control Protocol (TCP). The measurements were made using iperf [16] and nuttcp [17], which are well-known network benchmark tools. The purpose of this evaluation was to confirm the packet manipulation overhead, because extension or shortening of a header may result in a big overhead. Figure 3 shows the evaluation model, in which two hosts communicated 50 with each other through the developed kernel module. The virtual interface is constructed by tun during the measurements. In addition, throughput overhead may depend on MSS size, because the ratio of header size to total packet size will be larger when the MSS size decreases. Hence, we evaluated the throughput of certain sizes of MSS. From this evaluation, we were able to determine the packet manipulation overhead in the proposed implementation. Details of the evaluation parameters are given in Table I.

Figures 4 and 5 show the throughput performance as the size of the MSS changes. The results are an average of ten measurements and show that the throughput of the developed kernel module has almost the same level of performance as the general Linux kernel. The reason that the throughput of the IPv4 general Linux kernel is slightly better than that of the IPv6 general Linux kernel is the difference in total packet length due to the header sizes of IPv4 and IPv6. In addition, because the ratio of header size to total packet size increases as MSS size decreases, the deficit also increases for smaller MSS.

Figures 6 and 7 show the standard deviation of throughput performance as the size of the MSS changes. The results show that the performance of the developed kernel module is similar to that of the general Linux kernel. This means that the load of the developed kernel module does not affect the network performance.

Figure 8 shows the Round Trip Time (RTT) as the size of the MSS changes. The results show that the developed kernel

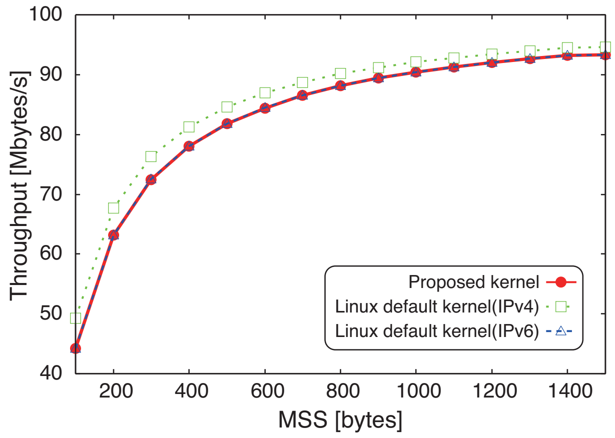

Figure 4. Throughput performance(iperf).

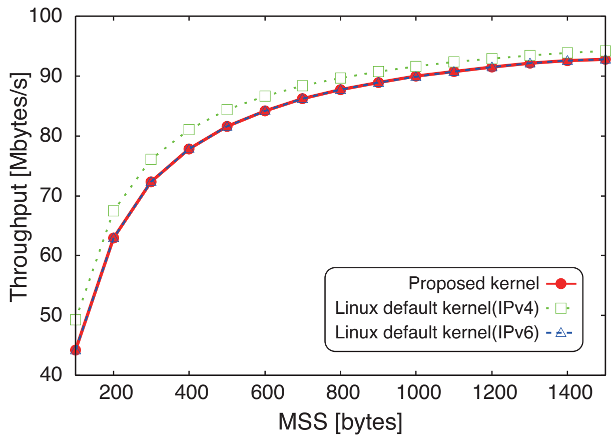

Figure 5. Throughput performance(nuttcp).

module has almost the same round trip time as the general Linux kernel. This means that the packet manipulation mechanism in the developed kernel module does not take much time and does not affect the transmission delay for communication.

These results indicate that using the Linux kernel module for netfilter can achieve high throughput and short processing delay.

VI. Conclusion

This paper presents a newly developed kernel module that performs IPv4/IPv6 address translation for IPv4-oriented applications. This kernel module provides virtual IPv4 addresses to IPv4-oriented applications, enabling them to communicate with IPv6 hosts through IPv6 networks. Since the kernel module can be implemented without modification of the general Linux kernel, it can easily be used to support IPv4 applications in IPv6 networks. The developed kernel module also supports an application level gateway for SIP messages. The packet manipulation of the proposed implementation takes place in Linux kernel space, thus achieving

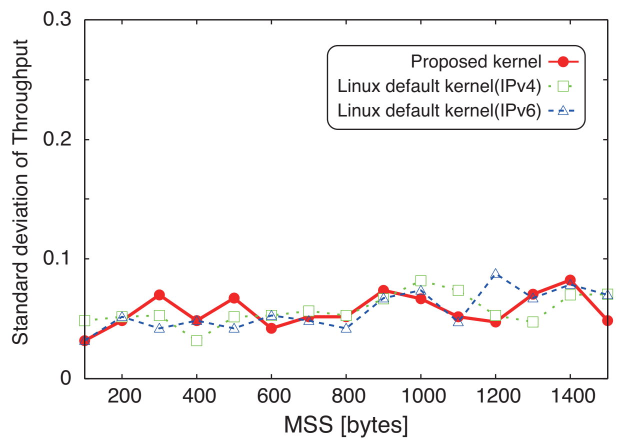

Figure 6. Standard deviation of throughput(iperf).

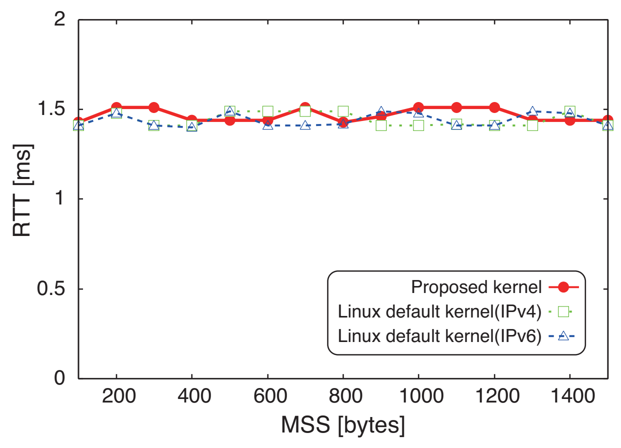

Figure 8. Round Trip Time(nuttcp).

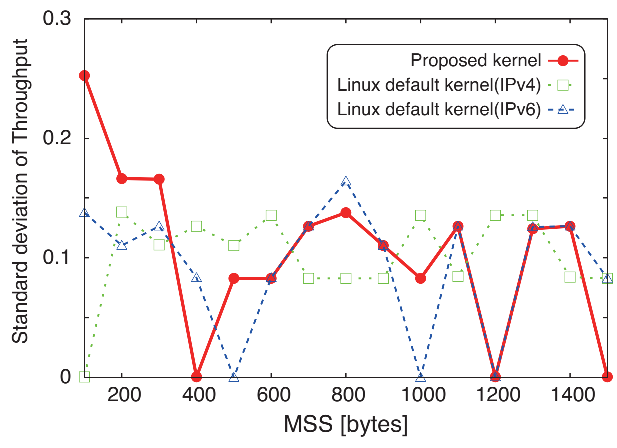

Figure 7. Standard deviation of throughput(nuttcp).

a high throughput performance and short processing delay by reducing memory copy in operating systems.

ACKNOWLEDGMENT

The authors thank Tokyo System House Co., LTD. for the valuable comments. A part of this research was supported by Japan Science and Technology Agency.

REFERENCES

[1] http://www.apnic.net/publications/news/2011/final-8, retrieved: Dec., 2011.

[2] D. Wing, "Network Address Translation: Extending the Internet Address Space," IEEE Internet Computing, Vol. 14, No. 4, pp. 66 – 70, July-Aug. 2010.

[3] E. Nordmark and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers," IETF RFC 4213, Oct. 2005.

[4] Y. Xia, B. S. Lee, C. K. Yeo, and V. L. S. Seng, "An IPv6 Translation Scheme for Small and Medium Scale Deployment," 2010 Second International Conference on Advances in Future Internet (AFIN), pp. 108 – 112, Jul. 2010.

[5] A. Durand, R. Droms, J. Woodyatt, and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion," IETF draft, May 2011.

[6] M. Bagnulo, P. Matthews, and I. van Beijnum, "NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," IETF draft, Mar. 2009.

[7] G. Tsirtsis and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)," IETF RFC 2766 Feb. 2000.

[8] F. Baker, X. Li, C. Bao, and K. Yin, "Framework for IPv4/IPv6 Translation," IETF draft Aug. 2010.

[9] C. Aoun and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status," IETF RFC 4966, Jul. 2007.

[10] K. Tsuchiya, H. Higuchi, and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)," IETF RFC 2767, Feb. 2000.

[11] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," IETF RFC 3261, Jun. 2002.

[12] W. Chen, C. Su, and J. Weng, "Development of IPv6-IPv4 translation mechanisms for SIP-based VoIP applications," 19th International Conference on Advanced Information Networking and Applications, AINA 2005. vol. 2 pp. 819 – 823, Mar. 2005.

[13] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," IETF RFC 5389, Oct. 2008.

[14] ISO/IEC 29341-1:2008, UPnP Device Architecture – Part 1: UPnP Device Architecture.

[15] http://www.netfilter.org, retrieved: Dec., 2011.

[16] http://iperf.sourceforge.net, retrieved: Dec., 2011.

[17] http://www.lcp.nrl.navy.mil/nuttcp, retrieved: Dec., 2011.