An Experimental Evaluation of IP4-IPV6 IVI Translation

Anthony K. Tsetse
Computer Information Systems, Livingstone College, Salisbury, NC 28147
asetse@livingstone.edu

Alexander L. Wijesinha, Ramesh Karne, Alae Loukili, Patrick Appiah-Kubi
Department of Computer & Information Sciences, Towson University, Towson, MD 21252
{awijesinha, rkarne, aloukili, appiahkubi}@towson.edu

ABSTRACT

While IPv6 deployment in the Internet continues to grow slowly at present, the imminent exhaustion of IPv4 addresses will encourage its increased use over the next several years. However, due to the predominance of IPv4 in the Internet, the transition to IPv6 is likely to take a long time. During the transition period, translation mechanisms will enable IPv6 hosts and IPv4 hosts to communicate with each other. For example, translation can be used when a server or application works with IPv4 but not with IPv6, and the effort or cost to modify the code is large. Stateless and stateful translation is the subject of several recent IETF RFCs. We evaluate performance of the new IVI translator, which is viewed as a design for stateless translation by conducting experiments in both LAN and Internet environments using a freely available Linux implementation of IVI. To study the impact of operating system overhead on IVI translation, we implemented the IVI translator on a bare PC that runs applications without an operating system or kernel. Our results based on internal timings in each system show that translating IPv4 packets into IPv6 packets is more expensive than the reverse, and that address mapping is the most expensive IVI operation. We also measured packets per second in the LAN, roundtrip times in the LAN and Internet, IVI overhead for various prefix sizes, TCP connection time, and the delay and throughput over the Internet for various files sizes. While both the Linux and bare PC implementations of IVI have low overhead, a modest performance gain is obtained due to using a bare PC.[1]

Categories and Subject Descriptors

C.2.2 [Computer-Communication Networks]: Network Protocols – *Applications, Protocol architecture;* **C.2.6 [Computer-Communication Networks]**: Internetworking – *Routers, Standard;* **C.4 [Performance of Systems]**: *Measurement techniques;* **D.4.4 [Operating Systems]**: Communications Management – *Network communication.*

General Terms

Measurement and Performance.

Keywords

IPv6; IVI Translation; IPv4-IPv6 transition; bare PC; Linux.

[1] Copyright is held by the authors. This work is based on an earlier work: RACS'12 Proceedings of the 2012 ACM Research in Applied Computation Symposium, Copyright 2012 ACM 978-1-4503-1492-3/12/10.
http://doi.acm.org/10.1145/2401603.2401646

1. INTRODUCTION

Internet Protocol version 6 (IPv6) [1] is the next generation IP protocol for the Internet. IPv6 was introduced to replace the existing Internet Protocol version 4 (IPv4). Although IPv6 provides the benefit of increased address space (16 byte addresses), its deployment in the Internet has been slow primarily due to the established and familiar base of IPv4. Interim measures such as NAT and CIDR have also served to conserve the limited IPv4 address space. However, it is expected that IPv6 usage will increase as the number of networks and devices (such as low power sensors) that require IP addresses continues to grow. IPv6 and IPv4 are not compatible due to using different header sizes and formats.

To enable IPv4 and IPv6 to co-exist during the transition period, several mechanisms have been proposed. Dual stack and tunneling are described in RFC 2893 [2], with an update in RFC 4213 [3] to reflect their usage in practice. Dual stack devices have implementations of both IPv4 and IPv6 protocol stacks running independently. This makes it possible for such devices to process both IPv4 and IPv6 packets. Tunneling allows IPv6 packets to be carried on IPv4 networks and vice versa. Address translation is a transition mechanism that allows communication between IPv6 devices and IPv4 devices by converting between packets of each type. Several RFCs dealing with stateless and stateful translation have been recently published by the IETF. The new IVI translator [4] is an implementation of stateless IPv4-IPv6 translation that can also support stateful packet translation (the abbreviation IVI is derived from the Roman numerals IV for 4 and VI for 6).

We evaluate the performance of IVI translation experimentally in a LAN and on the Internet. We use a freely available implementation on Linux, and our own implementation on a bare PC that has no operating system or kernel. This enables us to compare the overhead due to only the IVI translation process with the overhead measured using the Linux IVI translator.

The remainder of this paper is organized as follows. In Section 2, we briefly discuss related work. In Section 3, we describe IVI translation. In Section 4, we discuss Linux and bare PC implementations of an IVI translator. In Section 5, we present the results of LAN and Internet experiments to evaluate IVI translation. In Section 6, we give the conclusion.

2. RELATED WORK

Several techniques have been proposed for translating between IPv4 and IPv6 packets. One such technique, Network Address Translation–Protocol Translation (NAT-PT and NAPT-PT) [5], was defined in RFC 2766. It allowed a set of IPv6 hosts to share a single IPv4 address for IPv6 packets destined for IPv4 hosts, and also allowed mapping of transport identifiers of the IPv6 hosts. RFC 2766 has since been recommended for historical status (in

RFC 4966) [6] because of several issues. In RFC 3142 [7], the operation of IPv6-to-IPv4 transport relay translators (TRT) is discussed. TRT enables IPv6-only hosts to exchange two-way TCP or UDP traffic.

Recent work on IPv4-IPv6 translation is detailed in a group of several related RFCs. A mechanism enabling an IPv6 client to communicate with an IPv4 server (stateful NAT64) is described in RFC 6146 [8]. Translation of IP/ICMP headers in NAT64 is done based on the Stateless IP/ICMP Translation Algorithm (SIIT) of RFC 6145 [9]. NAT64 uses an address mapping algorithm discussed in RFC 6052 [10] for IPv4-IPv6 address mapping. RFC 6147 [11] specifies the DNS64 mechanism, which can be used with NAT64 to enable IPv6 clients to communicate with IPv4 servers using the fully qualified domain name of the server. Translation as a tool for IPv6/IPv4 coexistence along with eight translation scenarios are discussed in RFC 6144 [12]. These RFCs provide the background for the IVI translator and translation approach given in [4], whose performance is the focus of this study. Details concerning an IVI implementation for stateless and stateful packet translation, and its deployment, are provided in [13].

3. IVI TRANSLATION

IVI translation employs a prefix-specific and stateless address mapping scheme that enables IPv4 hosts to communicate with IPv6 hosts as described in RFC 6219 [4]. In this scheme, which is based on that given in [10], subsets of an ISP's IPv4 addresses are embedded in the ISP's IPv6 addresses. During translation, an address of one type (v4 or v6) is converted to the other type. To represent IPv4 addresses in IPv6, the ISP inserts its IPv4 address prefix after a unique IPv6 prefix consisting of PL bits, where PL=32 (4), 40 (5), 48 (6), 56 (7), 64 (8), or 96 (12) (the number in parenthesis is the number of bytes p=PL/8). The representation for p<12 is shown in Fig. 1, where $v4_p$ and $v4_s$ respectively denote the prefix and suffix of the IPv4 address and u is a byte of all 0s (the length of each field in bytes appears above it). For example, if p=5 bytes (i.e. PL=40 bits), the PREFIX, $v4_p$, u, $v4_s$, and SUFFIX fields consist of 5, 3, 1, 1, and 6 bytes respectively. If p=12, the PREFIX field is followed immediately by the entire IPv4 address (u is not present).

P	8-p	1	p-4	11-p
PREFIX	$v4_p$	u	$v4_s$	SUFFIX

Figure 1. Representing IPv4 ISP addresses in IPv6

Thus, an ISP with an IPv6/32 address can have a prefix of /40, where bits 32 through 39 are set to all ones. Similarly, /24 IPv4 addresses are translated into /64 IPv6 addresses. The suffixes of these IPv6 addresses are normally set to all zeros. So with this mapping scheme [4], an ISP with a /32 IPv6 address 2001:dba:: will translate the IPv4 address 202.38.97.205/24 into the equivalent IPv6 address 2001:dba:ffca:2661:cd::.

Translation of IPv4 and IPv6 headers is done as prescribed in [9] with the source and destination address translated according to the address mapping scheme described above. After translating the transport layer (TCP or UDP) and ICMP headers, their checksums are recomputed to reflect the changes in the IP header. The IP header mapping scheme specified in [4] enables an IPv6 header to be constructed from an IPv4 header, and conversely, in the following manner (in addition to converting source and destination IPv4 or IPv6 addresses into their IVI mapped equivalent addresses). To translate an IPv4 header into a IPv6 header, the IHL, identification, flags, offset, header checksum, and options are discarded; version (0x4), TOS, protocol, and TTL fields are mapped into version (0x6), traffic class, next header, and hop limit respectively; and the value of total length is decremented by 20 and assigned to payload length. Conversely, to translate an IPv6 header into an IPv4 header, flow label is discarded; version (0x6), traffic class, next header, and hop limit fields are mapped into version (0x4), TOS, protocol, and TTL respectively; the value of payload length is incremented by 20 and assigned to total length; IHL is set to 5; and the remaining fields in the IPv4 header are assigned in the usual way (such as generating a value for the identification field and computing the IPv4 checksum). For convenience, we include Tables 1 and 2 taken from [4], which summarize the key elements of header translation from IPv4 to IPv6 and from IPv6 to IPv4 respectively that were discussed above.

Table 1. IPv4-to-IPv6 Header Translation (from [4])

IPv4 Field	IPv6 Equivalent
Version (0x4)	Version (0x6)
IHL	Discarded
Type of Service	Traffic Class
Total Length	Payload Length = Total Length – 20
Identification	Discarded
Flags	Discarded
Offset	Discarded
TTL	Hop Limit
Protocol	Next Header
Header Checksum	Discarded
Source Address	IVI address mapping
Destination Address	IVI address mapping
Options	Discarded

Table 2. IPv6-to-IPv4 Header Translation (from [4])

IPv6 Field	IPv4 Equivalent
Version (0x6)	Version (0x4)
Traffic Class	Type of Service
Flow Label	Discarded
Payload Length	Total Length = Payload Length + 20
Next Header	Protocol
Hop Limit	TTL
Source Address	IVI address mapping
Destination Address	IVI address mapping
	IHL=5
	Header Checksum Recalculated

4. IMPLEMENTATION

4.1 Linux Implementation

The Linux implementation of the IVI translator [14] has two main components: a configuration utility and a translation utility. The configuration utility is used to setup the routes using *mroute* or *mroute6* commands. The translation utility is responsible for translating the packets. To run the IVI translator on Linux, a patch has to be applied to the Linux kernel source. Once the patch is applied, the Linux kernel can be configured for IVI translation after recompiling the kernel. Patching the kernel results in some kernel source files being changed, and four other files (mapping.c, proto_trans.c, mroute.c and mroute6.c) being added for configuration and translation.

The functions required for translation referred to in the discussion below are all in the file proto_trans.c. When an IPv4 packet is received, the function *mapping_address4to6* is invoked to map the IPv4 address to an IPv6 address. Based on the value of the protocol field in the received IPv4 header, *icmp4to6_trans*, *tcp4to6_trans* or *udp4to6_trans* is invoked to translate the appropriate protocol. Each of these methods calls a function that computes the appropriate higher layer protocol (ICMP, TCP, or UDP) checksum as well. Once the protocol translation is done, *iphdr6to4_trans* is called to translate the IP header. For translating IPv6 to IPv4 packets, a similar process is used.

4.2 Bare PC Implementation

In a bare PC, applications run without the support of any operating system or kernel. Bare PC applications are built based on the bare machine computing (BMC) paradigm, earlier called the dispersed operating system computing (DOSC) paradigm [15]. Bare PC applications include Web servers [16], email servers [17], SIP servers [18], and split protocol servers [19]. The first bare PC application with IPv6 and IPv4 capability was a softphone used for studying VoIP performance [20]. This softphone was built by adding IPv6 capability to the bare PC softphone application described in [21].

The IVI translator application that runs on a bare PC was built by adding IVI translation capability on top of lean bare PC versions of the IPv6 and IPv4 protocols [22, 23]. The bare PC translator application included the implementation of two Ethernet objects ETH0 and ETH1 that directly communicate with its two network interface cards (NICs), each connecting to one of the two IP networks. A key difference between the Linux and bare PC implementations is that the IP protocols, address mappings, and protocol mappings in the latter are implemented as part of a single object called the XLATEOBJ. The architecture of the bare PC IVI translator is shown in Fig. 2.

The MAIN and RECEIVE (RCV) tasks are the only tasks running in the bare PC IVI translator application. The MAIN task runs when the system is started and whenever the RCV task is not running. The RCV task is invoked from the MAIN task when a packet (Ethernet frame) arrives on either interface. The capability to send out router advertisements on the IPv6 interface of the bare PC IVI translator is implemented in the GW object as part of the MAIN task. In XLATEOBJ, the *xto4handler* and *xto6handler* methods respectively translate between IPv6 packets and IPv4 packets. To map the source and destination IPv6 addresses to the corresponding IPv4 addresses, the *xto4handler* invokes the *addr6to4* method.

The processing logic of the bare PC IVI translator is shown in Fig. 3. The translator initially checks the arriving packet to determine if it is an IPv6 or IPv4 packet. If the packet is an IPv6 packet, the payload protocol type is determined from the next header field, and the appropriate protocol checksum is computed. If the checksum is valid, the IPv6 packet is translated into an IPv4 packet and forwarded. If the packet is an IPv4 packet, the IP checksum is computed. If the checksum is valid, the packet is translated into an IPv6 packet based on the protocol type in the packet and forwarded after validating the higher layer protocol checksum as needed. If any checksum fails, the packet is dropped.

5. EXPERIMENTAL RESULTS

5.1 LAN Experiments

5.1.1 LAN Environment

Fig. 4 shows the LAN used for testing.

This LAN was used to generate TCP data. A similar LAN was used to generate UDP and ICMP data. In the figure, S1, S2, S3 and S4 are 100 Mbps Ethernet switches, and R1 and R2 are IPv4 and IPv6 routers respectively. Router R1 serves the IPv4 routing domain, which consists of an IPv4 client C4 and an IPv4 server SV4. On the IPv6 side, C6 is an IPv6 client and SV6 is an IPv6 server. C6 and SV6 obtain the necessary routing information from the IPv6 router R2. The IVI translator is represented by XT.

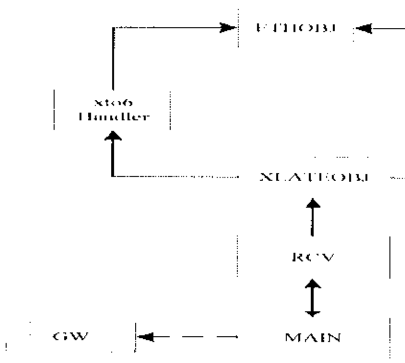


Figure 2. Bare PC IVI architecture

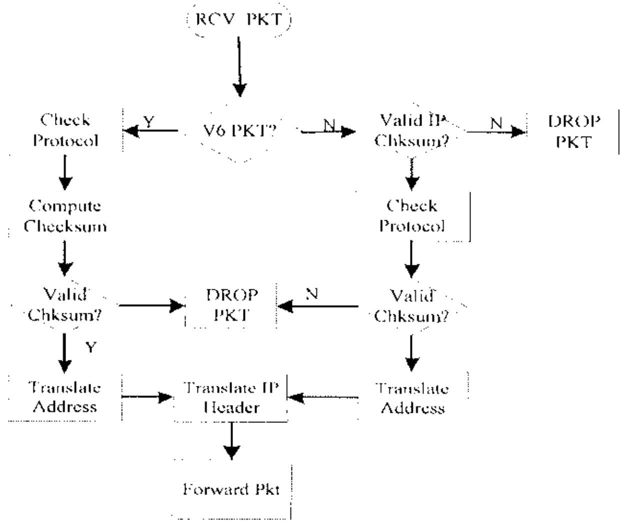


Figure 3. Bare PC IVI packet processing logic

The translator runs on a Dell Optiplex GX270 desktop with an Intel Pentium IV 2.4 GHz processor, 512 MB RAM, and Intel

PRO 10/100 and 3Com 10/100 NICs. The translator either runs with Fedora Linux (Fedora 12, Linux kernel 2.6.31) as the operating system, or as a bare PC application with no operating system or kernel. The client and server systems run on Dell Optiplex GX520s with a 2.6 GHz processor, 1 GB RAM, and an Intel PRO/1000 NIC, and Fedora Linux as the operating system. Apache HTTP Server 2.2.16 and Mozila Firefox v3.6.7 were used respectively as the Web server and Web browser.

To capture timings during translation with HTTP/TCP data, connections were made as follows. For IPv4 to IPv6 translation, the IPv4 client (C4) initiated a connection via the browser to the IPv6 Web server (SV6) in the IPv6 network. Router R1 forwarded the IPv4 request for a 320 KB file to the IVI translator XT, which translated the received packets to IPv6 packets, and sent them to the IPv6 Web server (SV6) via the IPv6 router R2. For IPv6 to IPv4 translation, the IPv6 client (C6) similarly initiated a connection via the browser to the IPv4 Web server (SV4). Router R2 forwarded this request to the IVI translator XT. The packet was translated to an IPv6 packet and forwarded to the IPv6 router R1, which delivered the packet to the IPv4 server (SV4).

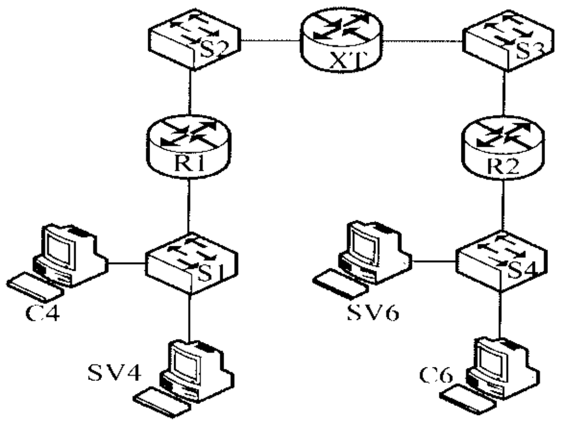


Figure 4. Test network for LAN Experiments (HTTP/TCP)

5.1.2 LAN Results

The internal timing data presented in this section was captured by inserting timing points in the Linux (Fedora) and the bare PC source code. The TCP traffic was generated by sending HTTP requests from clients to servers using the LAN in Fig. 4 as described earlier. Using a similar LAN, Mgen [24] was used to generate UDP traffic and the ping/ping6 utility was used to generate ICMP packets. Each experiment was run three times and the average data is reported (we omit the deviation since the differences in the values for each experiment were small).

Fig. 5 shows the time it takes internally to translate packets using the Linux implementation of the IVI translator. It is seen that the IP header translation is the most expensive operation for both IPv6-IPv4 and IPv4-IPv6 translation, while UDP, TCP or ICMP translation has almost the same translation time. The header translation time is larger since it includes both address and protocol (TCP, UDP or ICMP) translation.

It is also evident that the IP address translation time is larger than the UDP, TCP, and ICMP header translation time since the higher layer translations involve very little processing. Comparing translations between the IP versions in each direction, the IP address translation time is 3.9 µs and 0.5 µs higher than TCP, UDP, and ICMP translation time for the IPv4 to IPv6 and IPv6 to IPv4 translations respectively. The larger time for the IPv4 to IPv6 address translation compared to the reverse is because the IPv6 address needs to be constructed by extending the IPv4 address, whereas the IPv4 address is simply extracted from the IPv6 address.

Fig. 6 shows the corresponding IVI translation times on a bare PC. The times for IP header translation from IPv4 to IPv6 is 15.8 µs compared to 12 µs for translation from IPv6 to IPv4. TCP and ICMP translation take almost the same time: about 2.5 µs for IPv4 to IPv6 translation and 2.0 µs for IPv6 to IPv4 translation. For IP address mapping, the processing time is 5 µs from IPv4 to IPv6, and 2 µs from IPv6 to IPv4. For UDP, translating from IPv4 to IPv6 and from IPv6 to IPv4 takes 2 µs and 1.5 µs respectively.

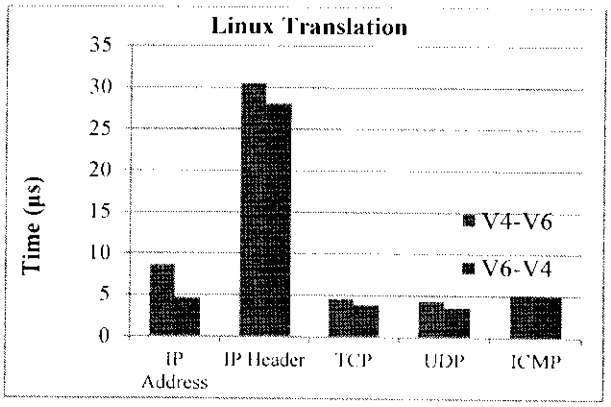


Figure 5. Linux translation overhead

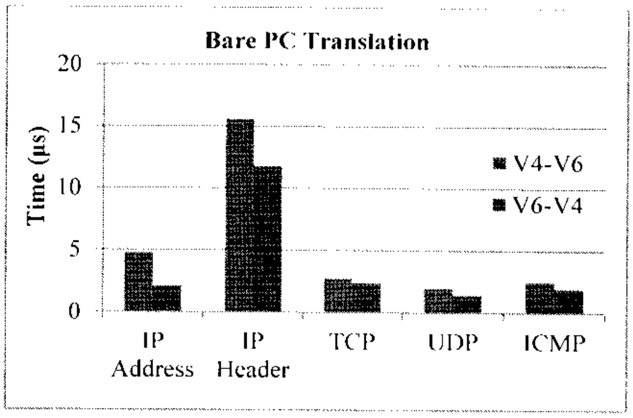


Figure 6. Bare PC translation overhead

Comparing the times for IVI translation from IPv4 to IPv6 for the Linux and bare PC implementations (using Figs. 5 and 6), it can be seen that 1) for TCP, UDP and ICMP translations, the time for the bare PC is about 2.3 µs less than for the Linux implementation; 2) for IP address translation, the processing time for the bare PC is about 3.9 µs less than for the Linux implementation; and 3) for IP header translation, the processing time for the bare PC is 15 µs less than for the Linux implementation.

Similarly, comparing the times for IVI translation from IPv6 to IPv4 translation for the Linux and bare PC implementations, it can also be seen that 4) for TCP and UDP traffic, the improvement in processing time due to using a bare PC is 2.2 μs; 5) for ICMP, the improvement in processing time is 3.2 μs; and 6) for IP address mapping, it is 2.6 μs.

To compare packets per second (pps) processed by the IVI translator for IPv4 to IPv6 translation and IPv6 to IPv4 translation, the Mgen generator was used to make TCP connections between source and sink. The results when transferring 1024-byte packets for which the throughput is at a maximum are shown in Fig. 7. As expected, pps for IPv4 to IPv6 translation is less than for IPv6 to IPv4 translation, which reflects the decrease in processing time for the latter; the drop in pps for IPv4 to IPv6 translation compared to IPv6 to IPv4 translation is about 18% for Linux and about 13% for the bare PC. The improvement due to using a bare PC instead of Linux is about 17% for IPv4 to IPv6 translation and about 9% for IPv6 to IPv4 translation.

We also compared the round trip times for 1024 byte ICMP Ping packets. Request packets undergo IPv4 to IPv6 translation and the reply packets undergo IPv6 to IPv4 translation. The results are shown in Fig. 8 and indicate that round trip time is reduced by about 17% on a bare PC.

These results show that the overhead for IVI translation for both the Linux and bare PC implementations is small. However, there is a small improvement in processing time (and hence, packets processed per second) when using a bare PC.

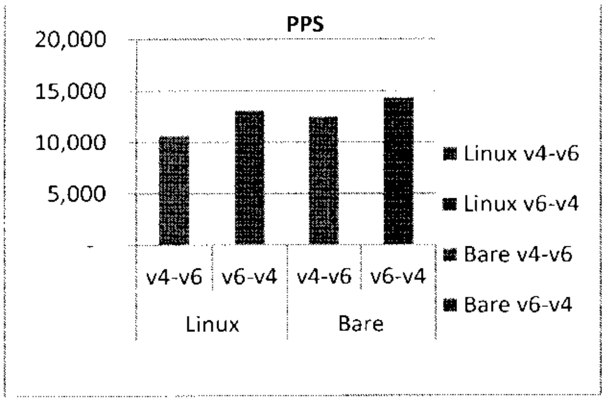


Figure 7. Packets per second (pps)

5.2 Internet Experiments

5.2.1 Internet Environment

Fig. 9 shows the network used to enable a client to connect to servers on the Internet via the IVI translator. One end of the test network with the client and translator was setup at Livingstone College (LC) in Salisbury, North Carolina. LC's network only allows IPv4 connectivity.

As shown in the figure, our IPv6 client (CL) connected to the LC network communicates with three different IPv4 Web servers (S1, S2, and S3) on the Internet. In accessing the Web servers, their IP addresses were used instead of the domain names since most ISPs on the Internet cannot resolve IVI mapped addresses. An HTTP request from the IPv6 client CL is initially routed to our default IPv6 router (RT6), which forwards the request to the IVI translator (XT). The IVI translator then translates the IPv6 packets to their equivalent IPv4 packets, and forwards the packets out on its IPv4 interface to LC's IPv4 border router LC4.These packets are seen as ordinary IPv4 packets, and the border router forwards them to the Internet. When LC4 receives the response from the Internet, it forwards the IPv4 packets to the IVI translator. The translator maps each packet to its IPv6 equivalent, and forwards it to the IPv6 router RT6, which delivers it to the IPv6 client CL. The client CL and the router RT6 run on Fedora Linux 18 kernel 3.6.7. The Linux IVI translator runs Fedora Linux 6. The same experiments were repeated with the bare PC IVI translator.

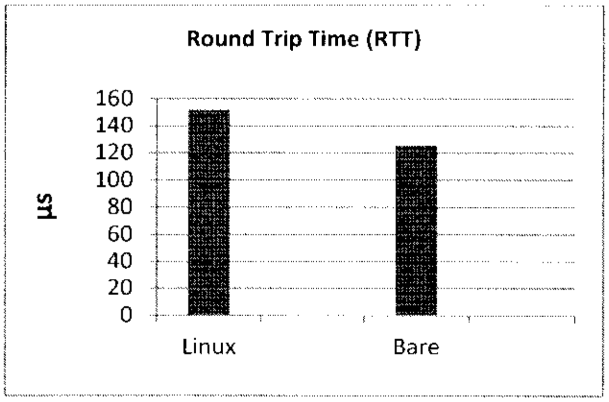


Figure 8. Ping round trip time (RTT)

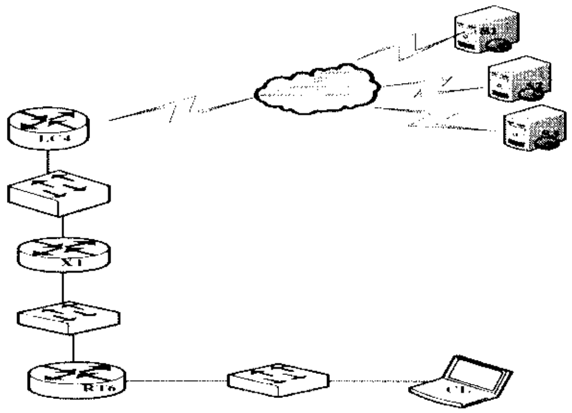


Figure 9. Test network for Internet experiments

5.2.2 Internet Results

Figs. 10 and 11 respectively show the IVI translation overhead using the Linux and bare PC IVI implementations when IPv4 addresses are mapped to IPv6 addresses with varying prefix lengths. We used prefix lengths ranging from 32 to 64 bits in increments of 8 bits with IPv4 addresses represented in IPv6 as shown in Fig. 1. It can be seen that translation overhead is lowest for a prefix length of 32 bits. With prefix lengths of 40 bits or more, the translation overhead is slightly higher for both the Linux and bare PC implementations. This can be explained by the fact that in mapping prefix lengths less than 40 bits as specified in [10], the IPv4 addresses appear within the IPv6 addresses in a contiguous manner, which results in relatively less processing overhead. Mapping IPv4 addresses to IPv6 addresses is more expensive than the reverse irrespective of whether a Linux or bare PC implementation is used.

Figs. 12 and 13 show the respective round trip times (RTTs) for Linux and bare translators by pinging three different Internet Web sites: www.itu.dk (Site 1), www.fh-offenburg.de (Site 2), and www.knust.edu.gh (Site 3) with different hop counts. We compare RTTs with IVI translation and for a direct request using IPv4. Requests to Site 1 have the highest RTT followed by Site 2 and Site 3. This is because the hop count for Site 1 is higher than for Site 2 and Site 3. It is also evident that the RTT when using IVI can be between a few milliseconds to about 15 milliseconds higher compared to a direct request to a site using IPv4. RTTs for Linux are higher than for the bare PC as expected. These results provide an estimate of the extra overhead due to IVI translation.

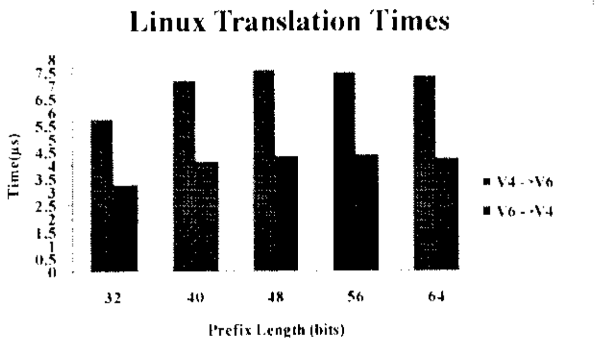


Figure 10. Linux translation with different prefix lengths

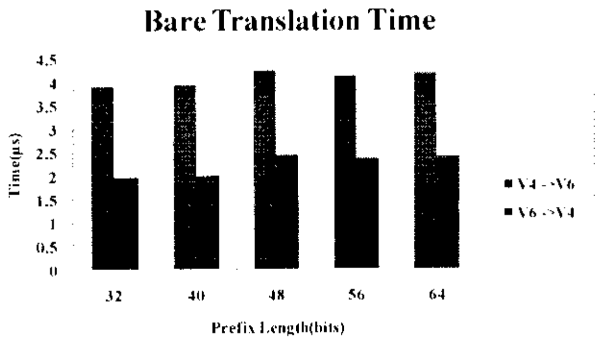


Figure 11. Bare PC translation with different prefix lengths

Figs. 14 and 15 show the respective connection times for Linux and bare translators using HTTP requests to the same three Web sites. Connection time is defined as the time for a TCP connection to be established. The connection times for Linux range from 42-121 ms for IPv4-IPv4 requests and from 46-126 ms for IVI translated requests. The corresponding connection times for the bare PC range from 40-118 ms for IPv4-IPv4 requests and from 44-124 ms for IVI translated requests.

Figs. 16 and 17 show the internal timings for the various IVI translation functions using the Linux and the bare PC implementations respectively with traffic to and from the Internet. It is seen that irrespective of the translation function and implementation, IPv4 to IPv6 translation has a higher overhead compared to IPv6 to IPv4 translation. The results for Internet traffic are very similar to those for LAN traffic (Figs. 5 and 6) as would be expected.

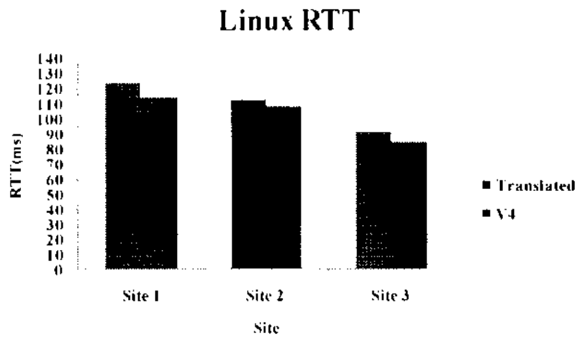


Figure 12. Linux RTTs for three Web sites

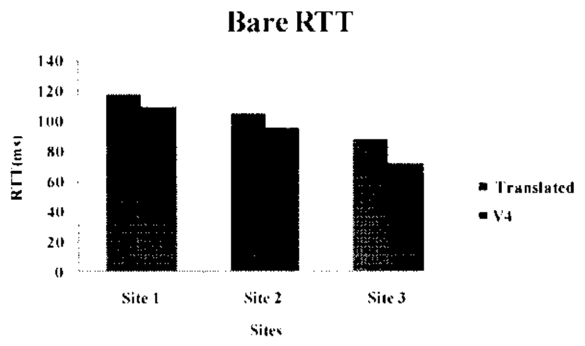


Figure 13. Bare PC RTTs for three Web sites

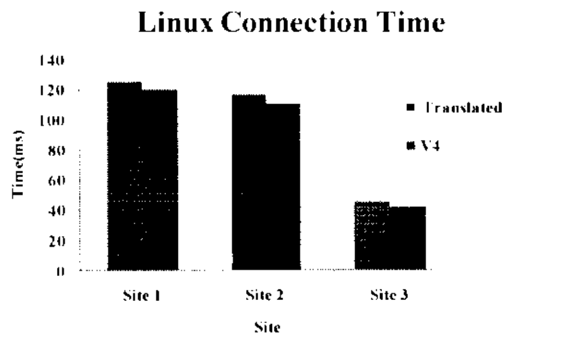


Figure 14. Linux connection time for three Web sites

We also conducted experiments over the Internet to evaluate IVI performance when the IPv6 LC client (in Livingston, North Carolina) makes HTTP requests for files of sizes 104, 318, and 531 KB respectively from an IPv4 Apache Web server running at Towson University (TU) in Towson, Maryland.

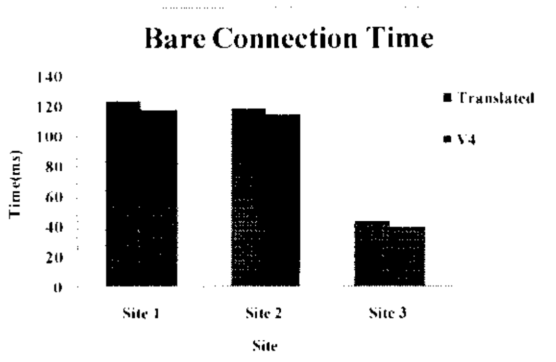


Figure 15. Bare PC connection time for three Web sites

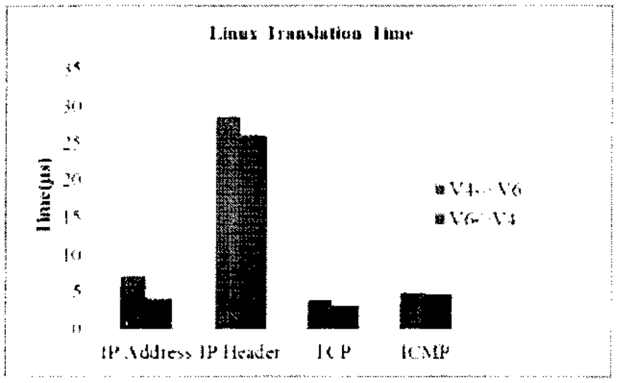


Figure 16. Linux Translation Time

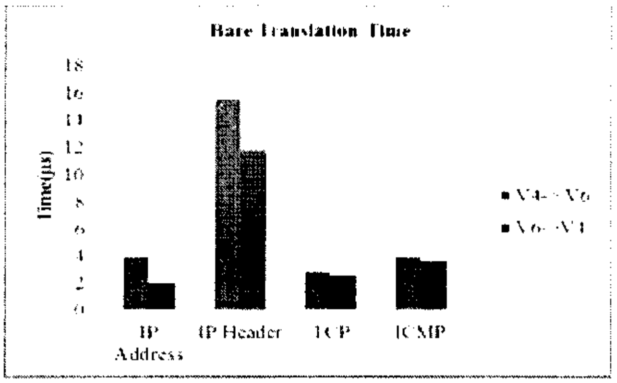


Figure 17. Bare Translation Time

Figs. 18-20 respectively compare the results for connection time, delay and throughput using the Linux and bare PC IVI implementations. The connection time is defined as for Figs. 14 and 15, and therefore is essentially the same for all three file sizes. Delay is defined as the time between the TCP SYN request and last ACK, and throughput is the data rate for the duration of the TCP connection. As expected, the larger files have the higher delays; they also have higher throughput. The throughput, which was obtained using a Wireshark packet analyzer [25] connected to the local LC switch, is the total throughput; it includes all the bytes (headers, data, and acks) seen by Wireshark. The measured throughput is therefore close to (but not identical to) the calculated throughput in Fig. 19, which is the file size divided by the delay. The performance difference between the bare PC and Linux IVI implementations for these Internet experiments is seen to be relatively small (as seen above for the case of the LAN experiments).

6. CONCLUSION

IVI translation is a technique recently proposed by the IETF to enable communication between hosts on IPv4 and IPv6 networks. We determined the overhead due to IVI translation by measuring the internal timings for both IPv6 to IPv4 and IPv4 to IPv6 translation on an ordinary desktop running Linux Fedora. We also implemented an IVI translator as a bare PC application on the same machine with no operating system or kernel, and compared the corresponding timings.

Several experiments were conducted in LAN and Internet environments to evaluate IVI translation performance. The results show that in general it is more expensive to translate packets from IPv4 to IPv6 than from IPv6 to IPv4 with a 4.1 μs difference on Linux and a 3.8 μs difference on a bare PC. It was found that address mapping is the most expensive operation in IVI translation regardless of the system on which the translator is implemented. IVI translation has little overhead for both the Linux and bare PC implementations regardless of the higher layer protocol carried in the payload. However, eliminating the operating system overhead would enable more efficient translation. The left-over CPU cycles could be used to process more packets, or for enhanced security-related processing during packet translation.

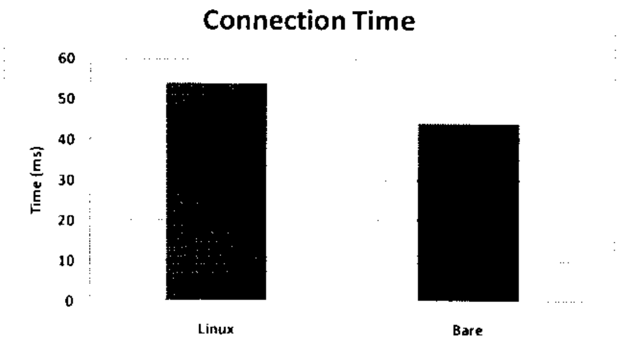


Figure 18. Connection time (LC to TU)

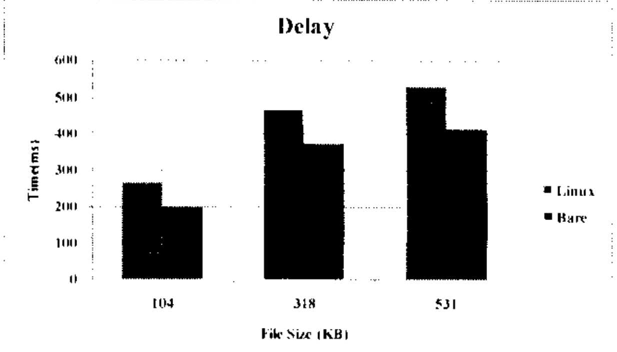


Figure 19. Delay for various files sizes (LC to TU)

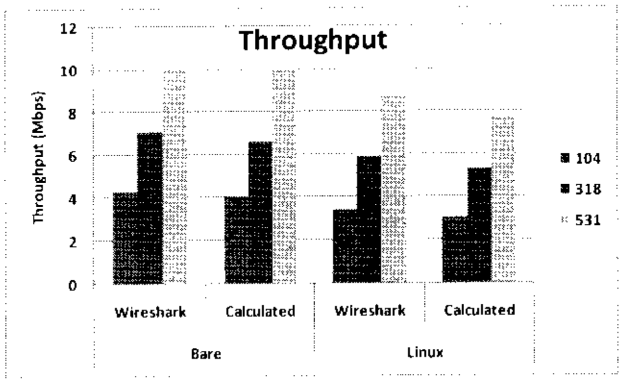


Figure 20. Throughput for various file sizes (LC to TU)

7. ACKNOWLEDGMENTS

We thank Bob Huskey for providing us with the code for his bare PC IPv4-IPv6 gateway implementation.

8. REFERENCES

[1] Deering, S., and Hinden, R. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, Dec. 1998.

[2] Gilligan, R., and Nordmark, E. Transition Mechanisms for IPv6 Hosts and Routers. RFC 2893, Aug. 2000.

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

[4] Li X., et al. The China Education and Research Network (CERNET) IVI Translation Design and Deployment for the IPv4/IPv6 Coexistence and Transition. RFC 6219, May 2011.

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

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

[7] Hagino, J., and Yamamoto, K. An IPv6-to-IPv4 Transport Relay Translator. RFC 3142, Jun. 2001.

[8] Bagnulo, M., Matthews, P., and van Beijnum, I. Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. RFC 6146, Apr. 2011.

[9] Li, X., Bao, C., and Baker, F. IP/ICMP Translation Algorithm. RFC 6145, Mar. 2011.

[10] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and Li, X. IPv6 Addressing of IPv4/IPv6 translators. RFC 6052, Oct. 2010.

[11] Bagnulo, M., Sullivan, A., Mathews, P., and van Beijnum, I. DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. RFC 6147, Apr. 2011.

[12] Baker, F., Li, X., Bao, C., and Yin, K. Framework for IPv4/IPv6 Translation. RFC 6144, Apr. 2011.

[13] Zhai, Y., Bao, C., and Li, X. Transition from IPv4 to IPv6: A Translation Approach. *Proc. Sixth IEEE International Conference on Networking, Architecture and Storage*, 30-39.

[14] Source Code of the IVI implementation for Linux: http://linux.ivi2.org/impl/, accessed 05/10/2012.

[15] Karne, R. K., Jaganathan, K. V., Rosa, N., and Ahmed, T. DOSC: dispersed operating system computing, *Companion to Proc. 20th annual ACM SIGPLAN Object Oriented Programming Systems and Applications Conference (OOPSLA '05)*, 2005, 55-62.

[16] He, L., Karne, R. K., and Wijesinha, A. L. The design and performance of a bare PC Web server. *International Journal of Computers and their Applications*, 15 (2):100–112, 2008.

[17] Ford, G. H., Karne, R. K., Wijesinha, A. L., and Appiah-Kubi, P. The design and implementation of a bare PC email server. *Proc. 33rd annual IEEE International Computer Software and Applications Conference (COMPSAC '09)*, 2009, 480-485.

[18] Alexander, A., Yasinovskyy, R., Wijesinha, A., and Karne, R. SIP server implementation and performance on a bare PC. *International Journal on Advances in Telecommunications*, 4 (1 and 2):82–92, 2011.

[19] Rawal, B. S., Karne, R. K., and Wijesinha, A. L. Splitting HTTP requests on two servers. *Proc. Third International Conference on Communication Systems and Networks (COMSNETS)*, 2011, 1-8.

[20] Yasinovskyy, R., Wijesinha, A. L., Karne, R. K., and Khaksari, G. A comparison of VoIP performance on IPv6 and IPv4 networks. *Proc. IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, 603-609, 2009.

[21] Khaksari, G. H., Wijesinha, A. L., Karne, R. K., He, L., and Girumala, S. A peer-to-peer bare PC VoIP application. *Proc. Fourth IEEE Consumer Communications and Networking Conference (CCNC)*, 2007, 803-807.

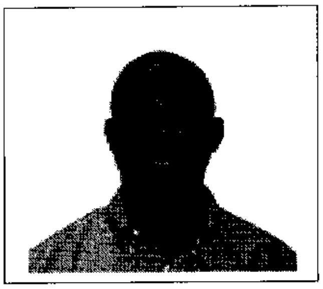
[22] Tsetse, A. K., Wijesinha, A. L., Karne, R. K., and Loukili, A. L. A 6to4 gateway with co-located NAT. *Proc. IEEE International Conference on Electro/Information Technology*, 2012, 1-6.

[23] Tsetse, A. K., Wijesinha, A. L., Karne, R. K., and Loukili, A. L. A bare PC NAT box. *Proc. International Conference on Communications and Information Technology*, 2012, 281-285.

[24] Mgen, http://cs.itd.nrl.navy.mil, accessed 05/10/2012.

[25] Wireshark, http://www.wireshark.org, accessed: 02/18/2013.

ABOUT THE AUTHORS:



Anthony K. Tsetse graduated from Towson University in 2012 with a doctorate in Information Technology. He received a B.S. in Computer Science from the Kwame Nkrumah University of Science and Technology, Kumasi, Ghana; an M.S. in Communication and Media Engineering from The Offenburg University of Applied Sciences, Germany; and an M.S.in Information Technology from the IT University of Copenhagen, Denmark. He is an Assistant Professor in the Department of Computer Information Systems at Livingstone College. His current research interests include bare machine computing, network performance analysis, cloud computing, wireless networks, and network security.

Alexander L. Wijesinha is a Professor in the Department of Computer and Information Sciences at Towson University. He holds a Ph.D. in Computer Science from the University of Maryland Baltimore County, and both an M.S. in Computer Science and a Ph.D. in Mathematics from the University of Florida. He received a B.S. in Mathematics from the University of Colombo, Sri Lanka. His research interests are in computer networks including wireless networks, VoIP, network protocol adaptation for bare machines, network performance, and network security.

Ramesh K. Karne is a Professor in the Department of Computer and Information Sciences at Towson University. He obtained his Ph.D. in Computer Science from the George Mason University. Prior to that, he worked with IBM at many locations in hardware, software, and architecture development for mainframes. He also worked at the Institute of Systems Research at University of Maryland, College Park as a research scientist. His research interests are in bare machine computing, networking, computer architecture, and operating systems.

Alae Loukili holds a doctorate in Information Technology and a Masters degree in Computer Science from Towson University. He also holds a Masters degree in Electrical Engineering from l' Ecole Nationale Supérieure d'Electricité et de Mécanique de Casablanca.He is currently a faculty member in the Department of Computer and Information Sciences at Towson University. His research interests are in computer networks and data communications, wireless communications, and bare machine computing.

Patrick Appiah-Kubi earned a doctorate in Information Technology from Towson University, an MS in Electronics and Computer Technology from Indiana State University, and a BS in Computer Science from Kwame Nkrumah University of Science and Technology, Ghana. He is a lecturer in the Computer and Information Sciences Department at Towson University. He previously worked as a system engineer and IT consultant at MVS Consulting in Washington DC. His research interests are in bare machine computing systems, networks, and security.