A 6to4 Gateway with Co-located NAT

Anthony K. Tsetse, Alexander L. Wijesinha, Ramesh K. Karne, and Alae Loukili
Department of Computer & Information Sciences
Towson University
Towson, MD 21252, U.S.A.
{atsetse, awijesinha, rkarne,aloukili}@towson.edu

Abstract—IPv6 was proposed as the next-generation IP primarily to deal with the problem of IPv4 address depletion caused by the rapid growth of the Internet. 6to4 tunneling is one of the currently used transition mechanisms for enabling IPv6 devices and networks to connect to today's Internet, which is primarily IPv4-based. Since most internal networks use private IPv4 addresses, it becomes necessary to provide both 6to4 and NAT functionality at the network boundary in order to handle IPv4 and IPv6 traffic. We evaluate the performance of a 6to4 Linux gateway with a co-located NAT. To enable 6to4 and NAT overhead to be determined, we also compare performance of the Linux gateway and a compatible 6to4 gateway with a co-located NAT that runs on a bare PC with no operating system or kernel installed. We describe the design and implementation of the bare PC 6to4 gateway with a co-located NAT, and also compare the performance of 6to4 in a test LAN with a co-located and a decoupled NAT (where 6to4 and NAT run on different devices). We conducted experiments with HTTP and VoIP traffic, and also measured RTT and CPU utilization. The results show that performance using 6to4 with a co-located NAT is better than with a decoupled NAT regardless of whether a Linux or a bare 6to4 gateway is used. In general, performance improvements with a co-located versus a decoupled NAT range from 34%-57% for the bare PC gateway and 7%-45% for the Linux gateway. Furthermore, performance improvements for a bare PC versus a Linux gateway range from 23%-86% with co-located and decoupled NATs. A 6to4 gateway with a co-located NAT can be used to improve network performance during the IPv6-IPv4 transition.

Keywords-IPv6; 6to4; gateway; tunelling; NAT; bare PC.

I. Introduction

Deployment of Internet Protocol Version 6 (IPv6) [1] in the Internet has been relatively slow since its introduction over a decade ago. There are a variety of business and practical reasons for the low prevalence of IPv6 networks. However, the difficulty of agreeing on a single technology or standard for use during the IPv4-IPv6 transition has made it harder for IPv6 networks to communicate across the existing IPv4 network infrastructure. Several transition mechanisms were originally proposed [2]. Since then, an automatic tunneling technique known as 6to4 [3] has become one of the most widely used transition mechanisms [4]. 6to4 can be deployed on end systems as well as on routers/gateways, and most operating systems support 6to4.

Unfortunately, using 6to4 on the Internet has proved to be challenging due to asymmetry in outbound and return addresses with long routing paths, firewalls, and other causes. This has resulted in a relatively high rate of connection failures being reported on many Web sites, and suggestions have been made to disable 6to4 altogether. A recent informational RFC [5] provides advice to ISPs, content providers and implementers regarding the avoidance of 6to4 failures. It is expected therefore that 6to4 will continue to be used during the transition period.

6to4 gateways are usually dual stack devices deployed at the edge of the network. It is convenient to configure 6to4 gateways to handle both IPv4 and IPv6 packets that may be generated by internal networks. Since internal IPv4 traffic almost always uses private (non-routable) addresses, Network Address Translation (NAT) [6] is then needed. This implies that a 6to4 gateway must have a co-located NAT [7], or else have NAT performed on another device. We refer to the latter approach as 6to4 with a decoupled NAT. While a co-located NAT is convenient to use, it has more overhead than a gateway that only handles 6to4 traffic. We conduct studies to evaluate the performance of a 6to4 gateway on Linux with a co-located NAT. In order to compare the overhead of only the 6to4 and NAT functions, we also implemented a 6to4 gateway with a co-located NAT as a bare PC application that runs with no operating system (OS) or kernel support.

In particular, we describe the design and implementation of the bare PC 6to4 gateway with a co-located NAT, and present the results of experiments conducted in a test LAN environment to compare the performance of the Linux and bare PC gateways with a co-located NAT. Our results show that both the Linux and bare PC 6to4 gateways with a co-located NAT perform better than their respective counterparts with a 6to4 gateway and a decoupled NAT. We also determine the extent of network performance improvement for HTTP and VoIP traffic when a bare PC 6to4 gateway with co-located and decoupled NATs are used instead of the Linux systems.

The remainder of this paper is organized as follows. In Section II, we briefly discuss related work. In Section III, we describe the design and implementation of the bare PC 6to4 gateway with a co-located NAT, and in Section IV we present the results of the performance study. The conclusion is given in Section V.

II. Related Work

Currently, 6to4 [3], Teredo [7, 8], tunnel-brokers [9], dual stack [10], and translation [11] are the main IPv4-IPv6 transition mechanisms in use. 6to4 [3] allows IPv6 nodes to communicate with each other over the global IPv4 infrastructure. This is done by using a 2002 address prefix for IPv6 hosts.

Teredo and its Linux version Miredo help nodes located behind NATs to gain IPv6 connectivity. However, Teredo/Miredo requires relays and/or servers, and is intended to be used as a last resort due to its encapsulation overhead. Tunnel brokers can use Tunnel Setup Protocol (TSP) to set up tunnel parameters and encapsulate IPv6 in IPv4 and vice-versa. Dual stack devices have implementations of both IPv4 and IPv6 protocols running independently; most operating systems support dual stacks. The dual stack approach only allows communication between like network nodes (i.e., IPv6-IPv6 and IPv4-IPv4). Translation mechanisms attempt to translate the IPv6 headers into IPv4 headers and vice-versa. The China Education and Research Network (CERNET) recently proposed a new technique for IPv4-IPv6 translation [12].

Most studies dealing with IPv4-IPv6 transition propose new transition mechanisms, or compare the performance of existing transition mechanisms. In [13], a hierarchical routing architecture to integrate both IPv4 and IPv6 networks is proposed. A performance study on configured tunnels and 6to4 mechanisms running on Linux is described in [14], while 6to4 and tunnel broker performance is compared in [15]. In [16], VoIP performance over IPv4 and IPv6 is evaluated using a bare PC softphone as a control to analyze the effect of OS overhead on VoIP call quality. VoIP performance with IPv6 and IPsec is studied in [17], and it is concluded that NAT and 6to4 processing (using a decoupled NAT) have little impact on VoIP quality in IPv6 networks.

Although there have been several studies on the performance of various IPv4/IPv6 transition mechanisms, this is the first study evaluating the performance of 6to4 gateways with a co-located NAT. Our work also differs from previous work in that we describe the design and implementation of a 6to4 gateway application with NAT on a bare PC that runs without an OS. The study in [16] uses a bare PC softphone to study VoIP performance, while this study uses a bare PC as a network gateway device. Moreover, we use IPv6 background traffic for our measurements as opposed to [17], which uses IPv4 background traffic and primarily focuses on VoIP performance with 6to4 and IPsec. In contrast, we consider network performance with both VoIP and HTTP traffic passing through the 6to4/NAT gateway. The study in [18] only deals with the design, implementation, and performance of a bare PC NAT box for an IPv4 network. It does not consider IPv6 or the design and implementation of a 6to4 gateway with a co-located NAT. Here, we also compare network performance using 6to4 with both co-located and decoupled NATs.

III. DESIGN AND IMPLEMENTATION

The self-supporting bare PC 6to4 gateway application has its own lean IPv6 and IPv4 stacks with direct interfaces to the hardware. There is no OS or kernel in the bare machine, and the application itself processes packets arriving from its two network interfaces. Ethernet Objects ETHOBJ1 and ETHOBJ2 were implemented to interface with the two NICs. Fig. 1 shows the software architecture of the bare PC 6to4 application.

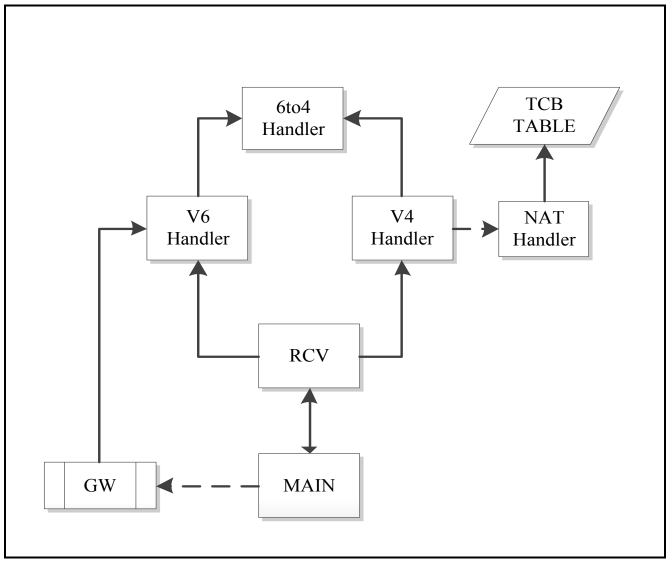

Figure 1. Bare PC 6to4 gateway software architecture

In the bare PC 6to4 gateway, only two tasks are implemented (MAIN and RCV). These two tasks are present in all bare PC applications. Other tasks are added as needed based on the application. The gateway application includes v4 and v6 handlers to process the IPv4 and IPv6 packets respectively. In addition, a 6to4 handler is used for 6to4 processing. The 6to4 gateway also has a NAT handler to process IPv4 traffic entering and exiting the internal network. The NAT mapping table is stored in a data structure called TCBTABLE. The bare gateway application also sends router advertisements to IPv6 clients as done by the radvd daemon [19] used in Linux routers. Router advertisements in the bare gateway are handled independently of the RCV task since they have to be sent even if no packets are received. All bare gateway application code is written in C/C++.

The bare PC 6to4 gateway application is started by invoking an executable boot program stored on a USB flash drive. The initial sector of the USB loads the menus that an administrator can use to select and configure the 6to4 application. Once the 6to4 executable is run, the self-supporting bare PC application object containing the monolithic executable is loaded. Control is then passed to the MAIN task, which starts the IPv6 router advertisements and passes control to the v6 handler. The RCV task is only called when a packet arrives on either interface.

The packet processing logic of the 6to4 gateway is shown in Fig. 2. When a packet arrives, the RCV task checks the packet to determine the packet type (i.e., IPv4 or IPv6). If the packet is an IPv6 packet, it is encapsulated and forwarded. If the packet is an IPv4 packet, the IP header is checked to see if it is a 6to4 packet. If so, the packet is forwarded after decapsulation or dropped, depending respectively on whether the IP checksum is valid or fails. If the packet is not 6to4, the usual IPv4 processing is done prior to forwarding it.

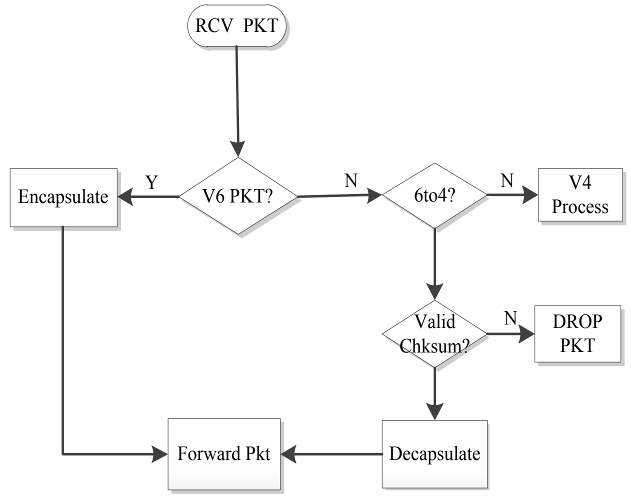

Figure 2. Packet processing logic

IV. PERFORMANCE

We performed several experiments to measure performance of 6to4/NAT gateways as described below. The generic network setup for the experiments is shown in Fig. 3. Modifications were made as needed to this setup for experiments with co-located or decoupled NATs. The 6to4/NAT gateways run on Linux (Fedora 12) or a bare PC. In the first set of experiments, 6to4 and NAT functionality were configured on the same gateway (co-located NAT). In this case, NAT was configured to process encapsulated IPv6 packets with protocol type 41. In the second set of experiments, NAT and 6to4 functionalities were decoupled to run on separate machines. We conducted four different experiments with decoupled NATs:

- Linux-Linux (both NAT and 6to4 on Linux)
- Linux-bare (NAT on Linux and 6to4 on bare)
- bare-Linux (NAT on bare and 6to4 on Linux)
- bare-bare (both NAT and 6to4 on bare)

The hardware and software used were as follows: the 6to4 and NAT gateways run on Dell OptiPlex GX270 PCs with Intel Pentium IV 2.4 GHz processors, Intel PRO 10/100 and 3Com 10/100 NICs, and 512 MB RAM. The Linux systems used Fedora 12 Linux kernel 2.6.35 with IPtables for NAT and radvd for router advertisements. The client and server systems were Dell Optiplex G520 PCs with 1 GB RAM, Intel PRO/1000 NICs, and 2.4GHz processors. The switches were 100 Mbps Ethernet Cisco Catalyst 2950 (S1 and S4) and Netgear GS108 (S2 and S3). Data collection was done by capturing packets using Wireshark [20] with mirrored ports on switches S1 and S4.

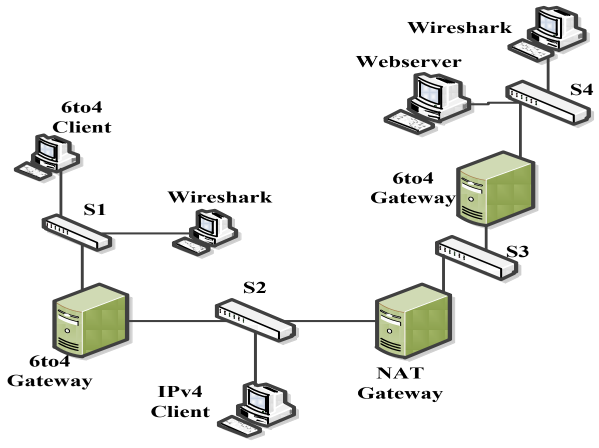

Figure 3. Experimental setup

The experiments were conducted by passing different types of traffic through the gateway in the presence of background traffic generated by using the MGEN traffic generator version 5 [21]. For HTTP traffic, we configured an Apache Web server with IPv6 and IPv4 running on Fedora 12. For VoIP traffic, we used Linphone clients [22]. Each experiment was run 3 times and the measured values were averaged. The values averaged did not differ significantly. The Linux gateway was configured with minimal functionality i.e., all unessential services on it were disabled.

A. Connection Time

We define the connection time as the time it takes for a TCP connection to be established by measuring the delay between the SYN and ACK from the client. The connections to the Web server were made in the presence of IPv6 background network traffic at 50 Mbps. Fig. 4 shows that the connection time for a decoupled NAT is higher than that for a co-located NAT. For Linux, the average connection times are 1.91 ms and 1.72 ms with decoupled and co-located NATs respectively, indicating an improvement of about 11% with a co-located NAT. For the bare PC, the average connections times are 1.38 ms and 0.98 ms with decoupled and co-located NATs respectively. This shows a performance improvement of 40% with a co-located NAT. Comparing the bare PC and Linux gateways, the bare PC performs 76% and 39% better than a Linux gateway with co-located and decoupled NATs respectively. With decoupled NATs, using a bare PC gateway for both devices (bare-bare) is best, and a bare 6to4 gateway and a Linux NAT (Linux-bare) has better performance than the reverse configuration (bare-Linux) as would be expected.

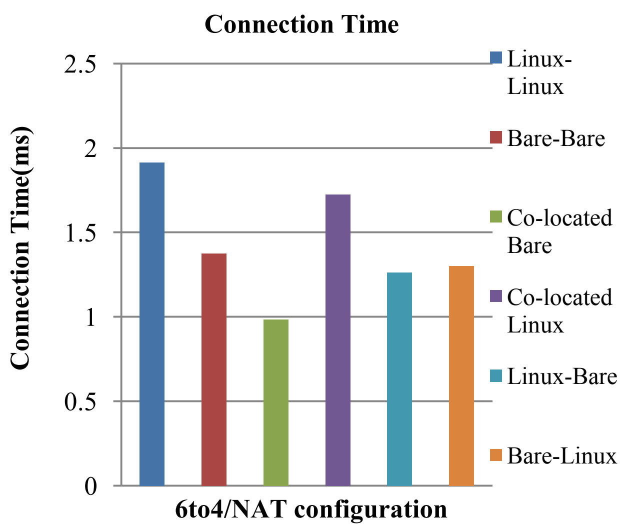

Figure 4. Connection time

B. *Response Time*

The response time is the time to download a file for an HTTP GET request i.e., the delay between the GET and OK messages. The HTTP traffic was generated by requesting HTML pages with embedded images of sizes ranging from 4 KB to 150 KB in the presence of 50 Mbps IPv6 background network traffic. Fig. 5 shows that performance of both the Linux and bare gateways with a co-located NAT is better than with a decoupled NAT. The performance improvements with a co-located NAT versus a decoupled NAT are 7% and 34% for the Linux and bare gateways respectively, and the bare PC performs 51% and 38% better than the Linux gateway with a co-located and decoupled NAT respectively. The figure also shows the increase in IPv6 response time as file size increases; with decoupled NATs, performance for a bare 6to4 gateway and a Linux NAT (Linux-bare) is again better than for the reverse configuration (bare-Linux).

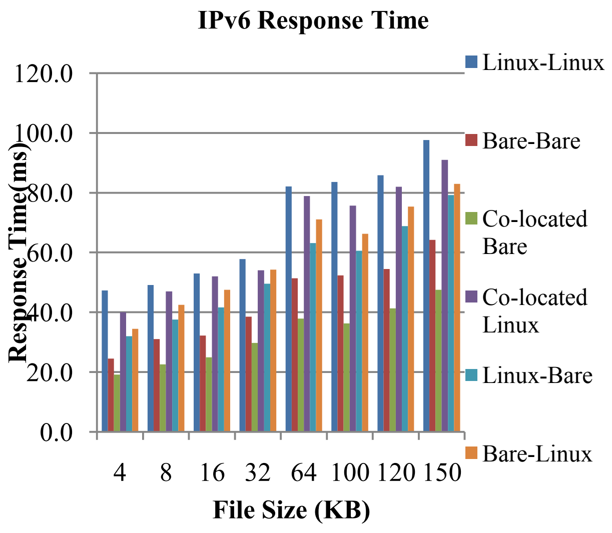

Figure 5. Response time

C. *VoIP Call Setup*

VoIP traffic was generated by playing a 3-minute audio file while the background traffic rate was varied from 0-90 Mbps. Fig. 6 shows the call setup times for VoIP at various background traffic rates measured as the delay between the SIP INVITE and the 200 OK messages. Call setup times with a co-located NAT is less than with a decoupled NAT for both the Linux and bare gateways. The improvements with co-located versus decoupled NATs are 38% and 57% for the Linux and bare gateways respectively, and the bare gateway has an 86% and 65% improvement over the Linux gateway with co-located and decoupled NATs respectively.

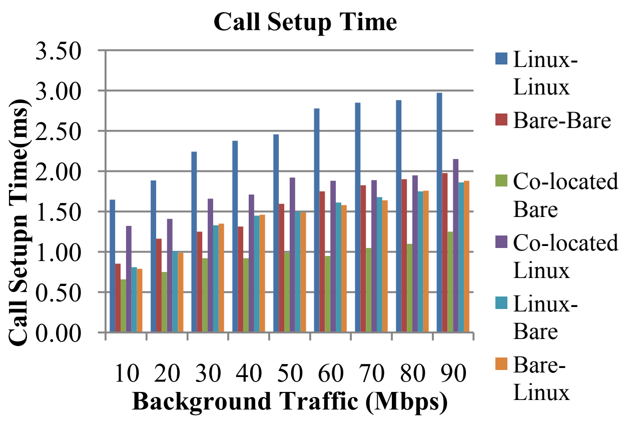

Figure 6. VoIP call setup time

D. *Mean Jitter*

Fig. 7 shows the increase in mean jitter for VoIP traffic with increasing background traffic. The performance improvements with a co-located NAT over a decoupled NAT for the Linux and bare gateways are 28% and 33% respectively, and the bare PC performs 82% and 75% better than the Linux gateway with co-located and decoupled NATs respectively.

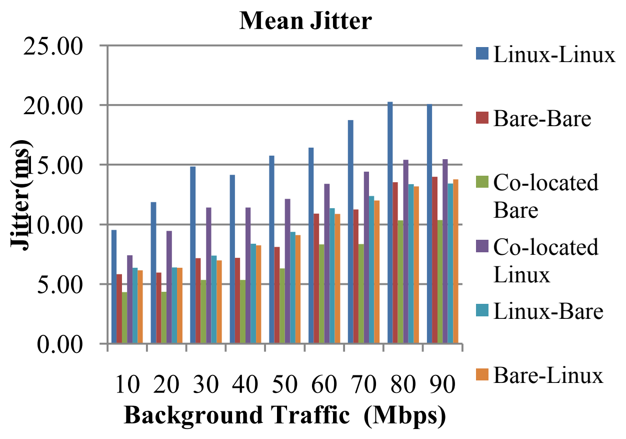

Figure 7. Mean jitter for VoIP

E. *VoIP Throughput*

Fig. 8 shows the decrease in VoIP throughput with increasing background traffic.

The performance improvements for the Linux and bare gateways with a co-located versus a decoupled NAT are 12% and 5% respectively, and the bare PC performs 23% and 35% better than the Linux gateway with co-located and decoupled NATs respectively.

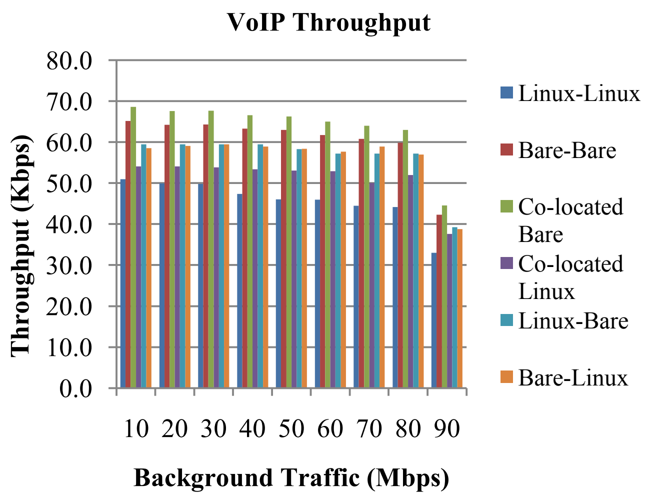

Figure 8. VoIP throughput

F. Packet Loss

The increase in packet loss as the background traffic rate increases is shown in Fig 9. No packet loss occurs when the background traffic is less than 50 Mbps. The packet loss ratio (or rate) is less with a co-located than with a decoupled NAT for both the bare and Linux gateways. The decrease in packet loss ratio with a co-located over a decoupled NAT for the Linux and bare gateways is 16% and 13% respectively, and the packet loss ratio decreases by 51% and 52% due to using a bare gateway instead of a Linux gateway with co-located and decoupled NATs respectively.

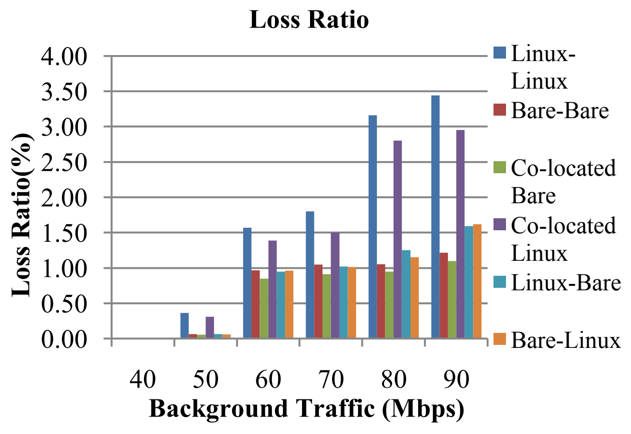

Figure 9. VoIP packet loss rate

G. RTT

Fig. 10 shows the IPv6 round trip time (RTT) for the network.

To measure the RTT, we used the ping6 utility to send 100 packets containing 1400 bytes each with background traffic varying from 10-90 Mbps. The RTT increases with increasing background traffic as expected irrespective of whether co-located or decoupled NATs are used, and a co-located NAT has a smaller RTT than a decoupled NAT for both the Linux and bare gateways. The RTT decreases by 45% and 34% with a co-located over a decoupled NAT for the Linux and bare gateways respectively, and the RTT decreases by 57% and 60% for the bare PC versus the Linux gateway with co-located and decoupled NATs respectively.

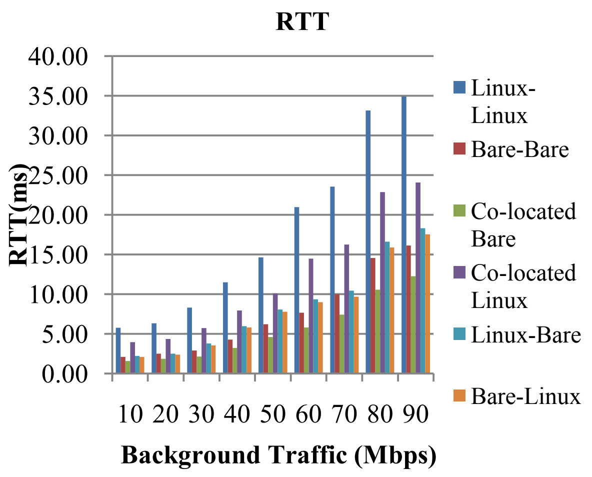

Figure 10. RTT for Ping6

H. CPU Utilization

To measure the CPU utilization of the co-located gateways, we varied the background traffic rate from 10-200 Mbps. Fig. 11 shows that the peak CPU utilization occurs with 90 Mbps of network traffic and is 1.9% for Linux and 0.5% for the bare gateway. The smaller CPU utilization in the bare gateway is due to using a single thread of execution and streamlined code to process packets.

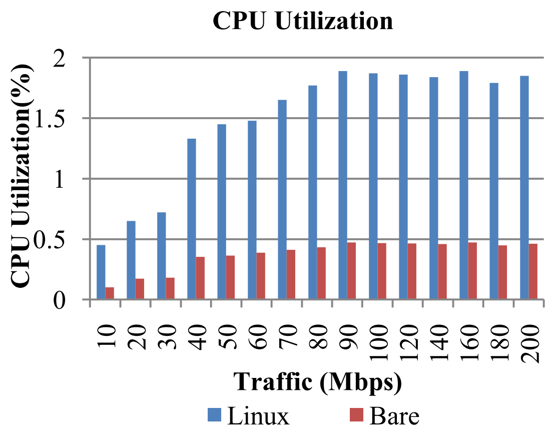

Figure 11. CPU utilization

V. CONCLUSION

We considered the performance of 6to4 gateways with co-located and decoupled NATs using Linux and a bare PC with no operating system. We also discussed implementation details of the latter. To compare gateway performance, we evaluated response and connection time for HTTP traffic, and call setup time, jitter, and throughput for VoIP traffic in a test LAN. We also compared packet loss, RTT, and CPU utilization. In general, our results show that performance of a 6to4 gateway is better with a co-located NAT than with a decoupled NAT. For a Linux gateway, there is a performance improvement of between 7%-45%, and for a bare PC gateway, there is a performance improvement of between 13%-57%. Moreover, using a bare PC instead of Linux gateway improves performance by 23%-86% with co-located and decoupled NATs. Reduction of operating system overhead can improve performance on a 6to4 gateway with either a co-located or a decoupled NAT.

REFERENCES

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

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

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

[4] D. Punithavathani and K. Sankaranarayanan, “IPv4/IPv6 Transition Mechanisms”, European Journal of Scientific Research ISSN 1450-216X Vol.34 No.1 (2009), pp.110-124.

[5] B. Carpenter, “Advisory Guidelines for 6to4 Deployment”,RFC 6343, August 2011.

[6] P. Srisuresh and K. Egevang: “Traditional IP network address translator (Traditional NAT)”, RFC3022, IETF Jan 2001.

[7] C. Huitema, “Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)”, RFC 4380, February 2006.

[8] D. Thaler, “Teredo Extension”, RFC 6081, January 2011.

[9] M. B. Viagenie and F. Parent, “IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP) ”, RFC 5572, February 2010

[10] E. Nordmark and R. Gilligan, “Transition Mechanisms for IPv6 Hosts and Routers”, RFC 4213, October 2005.

[11] G. Tsirtsis and P. Srisuresh, “Network Address Translation-Protocol Translation (NAT-PT)”, RFC2766, February 2000.

[12] X. Li 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.

[13] H. J. Liu and P. S. Liu, “Hierarchical Routing Architecture for Integrating IPv4 and IPv6 Networks”, 2008 IEEE Asia-Pacific Services Computing Conference.

[14] S. Narayan and S. Tauch, “IPv4-v6 Transition Mechanisms Network Performance Evaluation on Operating Systems”, 2nd International Conference on Signal Processing Systems (ICSPS), July 2010.

[15] C. Friaças, M. Baptista, M. Domingues and P. Ferreira, “6to4 versus Tunnel Brokers”, International Multi-Conference on Computing in the Global Information Technology (ICCGI'06).

[16] R. Yasinovskyy, A. L. Wijesinha, R. K. Karne and G. Khaksari, “A Comparison of VoIP Performance on IPv6 and IPv4 Networks”, ACS/IEEE International Conference on Computer Systems and Applications”, pp. 603-609 , May 2009.

[17] R. Yasinovskyy, A. L. Wijesinha, and R Karne, “Impact of IPsec and 6to4 on VoIP Quality over IPv6”, 10th International Conference on Telecommunications (ConTEL), pp 235-242, June 2009.

[18] A.K. Tsetse, A. L. Wijesinha, R. K. Karne, and A. Loukili, “A Bare PC NAT Box”, The Second International Conference on Communications and Information Technology (ICCIT 2012), to be published.

[19] Linux IPv6 Router Advertisement Daemon (radvd), http://www.litech.org/radvd/ accessed 12/30/2011.

[20] Wireshark packet analyzer, http://www.wireshark.org/,accessed 12/28/2011.

[21] Multi-Generator (MGEN), http://cs.itd.nrl.navy.mil/work/mgen, accessed 12/05/2011.

[22] Linphone, http://www.linphone.org/, accessed 12/28/2011.