Comparison of end-system IPv6 protocol stacks

S. Zeadally, R. Wasseem and I. Raicu

Abstract: The Internet Protocol version 6 (also known as IPv6) has been developed to replace the current IPv4 protocol. Various operating systems running at end-systems now support IPv6 protocol stacks and network infrastructures (hosts, routers) are currently being deployed to support IPv6 features. IPv6 stacks on end-systems constitute an important component in the migration and adoption of IPv6. To investigate the impact of IPv6 on user applications, an empirical evaluation was conducted on the performance of IPv4 and IPv6 protocol stacks on the Linux operating system for TCP and UDP protocols. The IPv6 performance obtained on Linux was compared with the performance of IPv6 of other commodity operating systems, namely Solaris 8 and Windows 2000. The experimental results demonstrate that the IPv6 protocol stack for Linux outperforms IPv6 stacks of these operating systems.

1 Introduction

The Internet Protocol version 4 (IPv4) [1] was developed in the early 1980s. Since then, it has established itself as a primary protocol which enables internetworking thereby allowing a vast array of client/server or peer-to-peer applications to communicate. TCP/IP engineers and designers recognised the need for upgrading in the late 1980s when it became apparent that the existing IP protocol would not be adequate to support the continued exponential growth of the Internet. In 1994, the Internet Engineering Steering Group approved a new Internet Protocol, first called IP next generation (IPng), and later known as Internet Protocol version 6 (IPv6) [2, 3]. The most important issue driving the need for IPv6, but not the only one, is the rapid depletion of IPv4 network addresses. IPv6 main features include: 'plug and play' which makes it easier for new users with not much TCP/IP knowledge to connect their machines to the network since all configuration will be done automatically, 'scalability' with its 128-bit address space, 'security' which includes encryption, 'real-time support' consideration (using the flow label field) and others, such as 'multicasting' [4].

Most of the existing protocol stacks, systems and applications run on IPv4-based systems. Changes to these systems can have significant impact on existing applications and must therefore be carefully implemented. While a principal design objective of IPv6 was to ease the transition from and coexistence with IPv4, the migration of IPv4-based systems to IPv6 will be a major challenge despite IPv6's built-in features that are backward-compatible with IPv4 [3]. Options, such as tunnelling of IPv4 packets over IPv6 and tunnelling IPv6 over IPv4, are needed for a smooth transition. In the last few years, network and operating system vendors have started to include support in many of their network applications and communication software products.

Migrating from IPv4 to IPv6 in current applications, or implementation of new IPv6 applications, requires the support of many components (application programming interfaces (APIs), protocol stack, routers etc.) of network systems. In this work however, we focus on the end-system which constitutes an important component in the deployment of IPv6-based systems. The IPv6 protocol stack at the end-system is expected to have a definite impact on end-to-end performance of emerging IPv6 applications. We have reported, in previous work [5], an extensive performance comparison between the IPv6 stacks running on Solaris and Windows 2000. Given the wide popularity and acceptance of the Linux operating system, we thought it would be interesting to carry the comparison obtained with Windows 2000 and Solaris further with the IPv6 stack performance delivered by Linux. Windows 2000 had the IPv4 stack as a standard protocol. However, to obtain IPv6 support, an add-on package was installed. There were two choices, both written by Microsoft and they were both in Beta testing. We chose the newer release of the two, 'Microsoft IPv6 Technology Preview for Windows 2000' [6], which is supported by Winsock 2 as its programming API. Solaris 8 and Linux both have dual production level IPv4/IPv6 stacks. The IPv6 protocol stack has been implemented in Linux kernel (version 2.1.8) since 1996 and current 2.2.x and 2.4.x kernel versions fully support IPv6. In this work we use Linux Red Hat version 7.3 (Valhalla) with kernel version 2.4.18-3.

2 Related works

The main motivation behind our work was driven by the fact that there are few published performance comparisons (as discussed below) between IPv4 and IPv6 protocol stacks at the end-system. This work differs from previous efforts of other researchers in that we performed a performance evaluation of three IPv6 protocol stack implementations on three widely used operating systems, in contrast to most previous works which compare IPv4 and IPv6 on one platform only or, as in the case of our previous work, on two operating systems [5]. We now briefly present some related work conducted recently and highlight how these

IEE Proceedings online no. 20040283

doi:10.1049/ip-com:20040283

Paper first received 16th April and in revised form 18th November 2003. Originally published online 21st May 2004

The authors are with the High-Speed Networking Laboratory, Department of Computer Science, Wayne State University, Detroit, MI, 48034, USA

efforts differ from ours. Draves *et al.* [7] presented a performance evaluation of a small subset of tests (actually only throughput) on a prototype IPv6 stack for Windows NT. In [8], a performance comparison was performed between IPv6 and IPv4 on Linux using a gigabit Ethernet adapter. The author conducted only some of the tests we report in this paper and did not compare IPv4/IPv6 stacks for different operating systems. In [9], the authors evaluate the performance of data transmission over IPv4 and IPv6 using various security protocols. They utilised end hosts with FreeBSD 2.2.8 and a KAME [10] IPv6 protocol stack and did not perform detailed testing based on the metrics discussed in this work. In [11], the author presented an evaluation of IPv6 compared to IPv4 using the dual stack implementation of KAME over a FreeBSD operating system using the ping utility and an FTP application; the metrics used were latency and file transfer throughput. They used the FTP application to find out the throughput rates over the IPv6 protocol, and used the ping utility to find the latency. They did not experiment with parameters such as packet size, connection time or protocol type (since they could not perform any UDP tests due to the nature of FTP).

3 Testbed configuration and measurement procedures

3.1 Testbed configuration

Two identical workstations were connected using a point-to-point (P2P) link. Using a P2P configuration eliminates many variables (e.g. router processing) from the experiments on the tested protocols on different operating systems using the same underlying hardware. Both workstations were each equipped with an Intel Pentium III 500 MHz processor, 256 megabytes of SDRAM PC100, 30GB IBM 7200 RPM IDE hard drives and 100 Mbit/s PCI Ethernet network adapters. The workstations were each loaded with Windows 2000 Professional, Solaris 8.0 and Linux Red Hat version 7.3 (Valhalla) (kernel version 2.4.18-3) operating systems on separate but identical hard drives.

3.2 Measurement procedures and performance metrics

We repeated similar experimental tests as these conducted in [5] and used the same performance metrics on the Linux platform which include: throughput (measured in Mbit/s), the rate at which bulk data transfers can be transmitted from one host to another over a sufficiently long period of time; round-trip time (RTT) (measured in microseconds), the amount of time it takes for one packet to travel from one host to another and back to the originating host (it is worthwhile noting that for TCP throughput and round-trip latency measurements we enabled the TCP_NODELAY option which has the effect of disabling the Nagle algorithm for send coalescing); socket creation time and TCP connection time, the amount of time (measured in microseconds) it takes to create a socket and to make a connection (for TCP), respectively; and web client/server simulation (measured in number of connections performed per second), this simulation tested how many connections could be performed per second. For each test, a socket was created, a TCP connection was set up, a one-byte message was sent and received, the connection was then torn down and the socket destroyed.

We implemented all performance measurement software used in our tests and for some tests we also used Netperf3 [12]. Most tests were executed for a sufficiently long period of time using various packet sizes ranging from 64 to 1408 bytes. The reason for this choice for the packet size range stems from observations that most packet sizes observed on networks and the Internet are within this range [13, 14].

4 Experimental results

In this Section we present and discuss the experimental results obtained from our tests for the IPv6 stacks on the three operating system platforms.

4.1 Throughput

Figures 1 and 2 show the TCP and UDP throughput results for different packet sizes. From the TCP throughput results in Fig. 1, we observe that for Solaris, there is a significant difference in throughput between IPv4 and IPv6 for message sizes less than 256 bytes. Actually, IPv4 yields almost three times higher throughput than IPv6 for these message sizes. However, the throughput differences decrease with increasing message sizes as shown in Fig. 1 (from 1024 bytes onwards). In the case of Windows 2000, throughput results are quite different. In this case, we observe that for small message sizes, we obtain very close throughput for both IPv4 and IPv6. However, above 512-byte messages, we note that IPv4 yields about 11% higher throughput than IPv6. In contrast, throughput for Linux is consistently high for all message sizes tested (around 93 Mbit/s) for both IPv4 and IPv6. There is only a minor throughput degradation of about 1 Mbit/s with IPv6 on Linux, with one exception at a message size of 256 bytes, where it drops to 70 Mbit/s for IPv6 compared to 94 Mbit/s with IPv4.

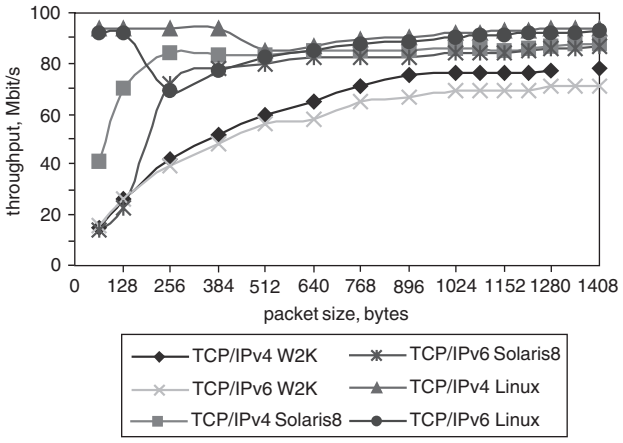

Fig. 1 *TCP throughput for IPv4 and IPv6 over Windows 2000, Solaris 8 and Linux for packet sizes ranging from 64 to 1408 bytes*

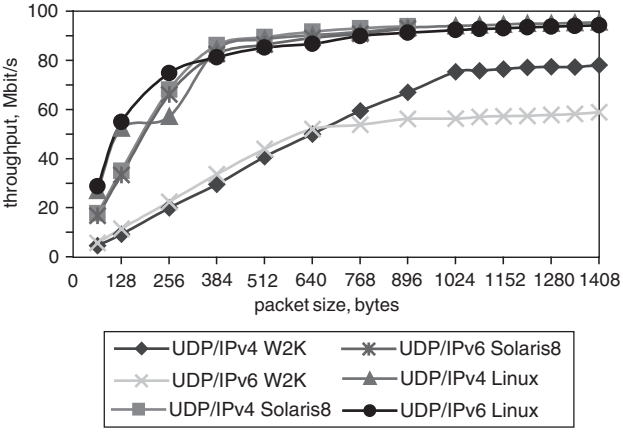

Fig. 2 *UDP throughput for IPv4 and IPv6 over Windows 2000, Solaris 8 and Linux with packet sizes ranging from 64 to 1408 bytes*

For UDP throughput (Fig. 2), the results obtained are quite different. In this case, for Solaris, we observe similar performance for both IPv4 and IPv6 even for small message sizes (less than 256 kbytes) compared to the performance obtained with TCP (Fig. 1). Similarly, on Windows 2000, we obtained similar throughput for both IPv4 and IPv6 for small messages. However, as message size increases, we observe a lower throughput for IPv6. Actually, beyond 1 kbyte message size, IPv6 results in 25% lower throughput than that obtained with IPv4. UDP throughput on Linux was close to those obtained on Solaris and shows similar trends for IPv4 and IPv6.

A plausible speculation that could explain the reason why we obtain better IPv4 and IPv6 performance for both TCP and UDP protocols on the UNIX platforms (Solaris and Linux) compared to Windows 2000 is probably due to the way kernel network buffers are allocated and used by Solaris and Linux operating systems. On Solaris, the STREAMS [15] subsystem is used in the implementation of its transport and network protocol stacks and network buffers of different sizes are pre-allocated by the kernel. The STREAMS subsystem uses a best-fit strategy for data buffer requests. When a network application transmits data, the underlying STREAMS subsystem uses one or more of the pre-allocated buffers (thereby avoiding overheads associated with buffer allocations). In the case of Linux, which is based on the traditional BSD socket implementation, we also have pre-allocation of a number of fixed-sized memory buffers called 'mbufs'. A typical mbuf size is around 256 bytes (depending on the underlying operating system). When a network application transmits network data larger than the mbuf size, a series of mbufs, chained together is used; the exact number of mbufs depends on some defined threshold value (typically around 900 bytes). When the application's data exceeds this threshold, the first mbuf is then chained to a larger memory buffer called a 'cluster' (cluster sizes also vary among operating systems). We hypothesise that pre-allocated kernel buffers (for network transmission) on both Solaris and Linux probably account for the close throughput performance obtained on these platforms for IPv4 and IPv6 for message sizes greater than 512 bytes. However, for smaller message sizes, we do observe lower throughput for IPv6, probably because the overheads of the IPv6 address size is more significant. For instance, for a message size of 128 bytes, overheads due to the address size is 12.5% and 3% for IPv6 and IPv4, respectively, whereas for a message size of 512 bytes, the overheads drop to 3% and 0.78% for IPv6 and IPv4, respectively.

We further speculate (in the absence of Windows 2000 source code availability) that kernel buffer allocation strategies are less efficient for the Windows platform compared to its UNIX counterparts. The inefficient buffer allocation strategies during network transmission probably explain why, with increasing message size, throughput (for both IPv4 and IPv6) is lowest for Windows 2000 since buffer allocation overheads probably increase with increasing message sizes.

4.2 Round-trip latency

Latency is an important performance metric for many network-based applications. Continuous media applications, such as those involving audio and video, are particularly sensitive to delay. Transactional applications (involving mostly request–reply operations), such as HTTP and DNS implementations, are sensitive to round-trip latency. In this context, in this Section, we present the impact of IPv6 and IPv4 stacks on end-to-end latency performance of user applications. The TCP and UDP round-trip latency results are shown in Figs. 3 and 4. As Fig. 3, shows, on Windows 2000, the TCP round-trip latency with IPv6 is about 30% higher (worse) compared to the IPv4 stack for small messages (up to 1 kbyte). In the case of Solaris, we observe a 5% increase in latency for IPv6 compared to IPv4. For packet sizes greater than 1 kbyte, on both Solaris and Windows 2000, we obtain around 1–2% increase in latency using IPv6 with TCP, most probably due to the amortisation of overheads associated with larger packet sizes (larger user payloads).

For the UDP latency results depicted in Fig. 4, we also obtain around 30% higher latency with IPv6 on Windows 2000 for messages up to 1 kbyte. With increasing message sizes, the latency difference between IPv4 and IPv6 packets decreases. In the case of Solaris, we also obtain around 5% higher latency with IPv6 compared to IPv4. It is interesting to note also that we do not observe significant latency differences between TCP and UDP (at least for the message sizes tested) for both Windows 2000 and Solaris operating systems despite the fact that TCP has more overheads associated with it compared to UDP.

From Fig. 3, the round-trip latency results of TCP/IPv6 for Linux were consistently lower than those on Solaris or Windows 2000 for both IPv4 and IPv6. In fact, for message

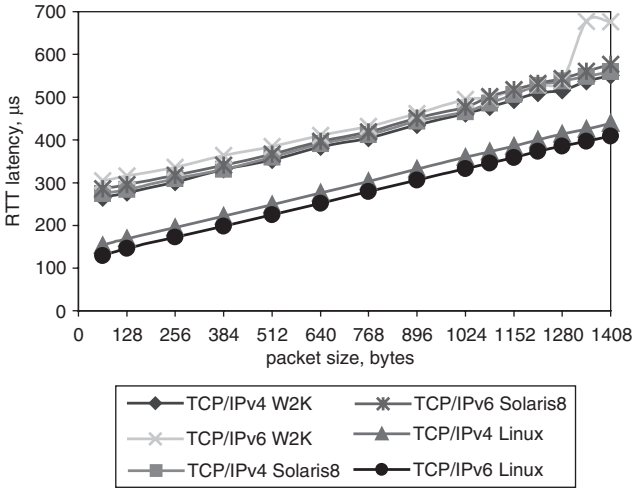

Fig. 3 *TCP latency results for IPv4 and IPv6 over Windows 2000, Solaris 8 and Linux with packet size ranging from 64 to 1408 bytes*

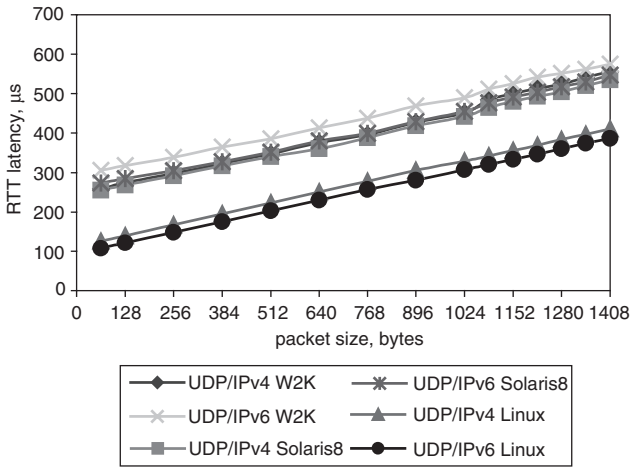

Fig. 4 *UDP latency results for IPv4 and IPv6 over Windows 2000, Solaris 8 and Linux with packet size ranging from 64 to 1408 bytes*

sizes up to 256 bytes, the latency results obtained with Linux are 50 lower than those obtained on Solaris and Windows 2000. For large message sizes (above 256 bytes), Linux gives around 30% better (i.e. smaller) latency than Solaris and Windows 2000. For UDP/IPv6, we obtained around 50–60% lower latency (messages up to 256 bytes) for Linux compared to Solaris and Windows 2000. For larger messages, Linux gives around 30% and 40% lower latency than Solaris and Windows 2000, respectively. It is also interesting to observe from Figs. 3 and 4 that both TCP and UDP yield around 7–15% lower latencies with IPv6 compared to IPv4. This result is different from the IPv4/IPv6 trends on Solaris and Windows 2000, where IPv6 latencies are always higher than IPv4.

It is worth commenting on two major results from Fig. 3 and 4 which include: (*a*) lower latencies with both IPv4 and IPv6 for Linux compared to those on Solaris and Windows 2000; and (b) lower latency for IPv6 than IPv4 for the Linux platform. Interestingly, Jobst and Feyrer [16] also concluded that IPv6 yields better performance than IPv4 for Linux compared to other operating systems (NetBSD 1.6 and Windows XP), which resulted in lower latency for IPv4 than IPv6. For us to be able to offer some plausible explanations for the two results mentioned in (*a*) and (*b*), we need to understand the origin of IPv6 in Linux, the first IPv6 network code was incorporated into the Linux kernel 2.1.8 in November 1996 and the code was based on the BSD application programming interface. Today, the UniverSAl playGround for Ipv6 (USAGI) [17] project works to deliver production quality IPv6 protocol stack for Linux. The Linux IPv6 stack code is heavily derived from the IPv6 code developed by the KAME [10] project: a joint effort of companies in Japan to provide free IPv6 stack for BSD variants (FreeBSD, OpenBSD, NetBSD). Thus, we believe that the design and implementation of the high-performance IPv6 stack resulting from the KAME project has also led to the Linux IPv6 stack performing better than the IPv6 stacks of Solaris and Windows 2000. In fact, this speculation is confirmed by Jobst and Feyrer [16], who found through their experimental tests that IPv6 performance on the NetBSD operating system was similar to the IPv6 performance obtained on Linux. We think this is a plausible explanation for the observation mentioned in (*a*) above. As far as the second observation mentioned in (*b*) is concerned, we speculate that the better (i.e. lower latency) performance of IPv6 over IPv4 is also partly due to the code derived from KAME. In addition, we also speculate that another possible reason for the lower latency with IPv6 may be because the IPv6 neighbourhood discovery mechanism is faster than the IPv4 ARP mechanism on Linux. We plan to do more testing and code instrumentation of the IPv6 stack in Linux to verify these hypotheses.

4.3 Socket creation time and TCP connection time

Figure 5 illustrates the TCP/UDP socket creation times and the time to set up a TCP connection on Windows 2000, Solaris, and Linux. It is clear from the results that Linux outperforms both Solaris 8.0 and Windows 2000 for TCP/UDP socket creation time and TCP connection time for both IPv4 and IPv6. Linux gives a sixteen-fold and nineteen-fold improvement (i.e. lower) over Windows 2000 in socket creation time for TCP/IPv4 and TCP/IPv6, respectively. However, Linux results in a four-fold decrease in socket creation time compared to Solaris for TCP/IPv4 and TCP/IPv6. It worthwhile pointing out that for TCP/IPv6 we observe an increase of 12%, 13% and 31% in socket creation times for Linux, Solaris and Windows 2000,

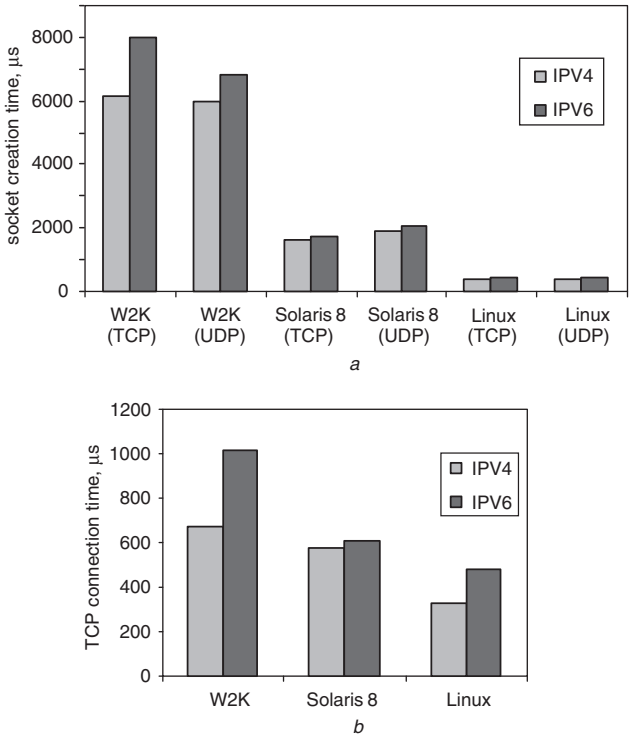

Fig. 5 *TCP and UDP socket creation time TCP connection time on Windows 2000 (W2K), Solaris and Linux for both IPv4 and IPv6*
a Socket creation time
b TCP connection time

respectively. We hypothesise that the difference in the implementation of sockets between Windows 2000 and the UNIX operating systems (Solaris and Linux) is probably the main reason which accounts for the large difference. The implementation factors responsible for the lower socket creation time for the UNIX variants (Solaris and Linux) are probably due to a more efficient BSD socket library than the socket library (actually called WS2_32.DLL) on Windows 2000, faster user/kernel switches during system calls and higher performance protocol stack implementations (TCP/UDP/IPv4/IPv6) on UNIX platforms (in this case Solaris and Linux) than on Windows 2000.

The TCP connection times are also higher for IPv6 on all three operating systems. Figure 5 shows that Linux gives the lowest connection times for IPv4 and IPv6. The connection time with IPv6 is around 50% higher than IPv4 for Windows 2000 and Linux. However, connection time for IPv6 on Solaris is only around 6% higher than IPv4. The increase in connection time for IPv6 is mostly likely due to the overhead caused by the increase in header size for IPv6 and address size (128-bit IPv6 address compared to the 32-bit IPv4 address) when a connection is set up. UDP does not use a connection mechanism like TCP, so we do not report its connection time.

4.4 Web client/server simulation

In recent years, we have witnessed the proliferation of a vast number of web servers. Web servers need to handle many transactions per second. These transactions are typically of short duration, and involve operations that are mostly request–reply in nature. Each of these operations basically requires a connection setup, performing a given data transfer, followed by closing down the connection. As a result, the performance delivered to web clients depends

partly on how fast these operations can be executed. We foresee that in the near future many web servers supporting dual IPv4 and IPv6 stacks will emerge. We were interested to explore the performance penalty, if any, for IPv6 web servers compared to IPv4 web servers. On Windows 2000, we obtained 147 and 115 connections per second for IPv4 and IPv6, respectively. We obtained 430 and 404 connections per second for IPv4 and IPv6, respectively, on Solaris 8 and 450 (IPv4) and 426 (IPv6) connections second on Linux. Solaris and Linux give almost similar performances but can support around four times more IPv6 connections than Windows 2000. The degradations from IPv4 to IPv6 are 5%, 6% and 22% for Linux, Solaris and Windows 2000, respectively. The more pronounced degradation for Windows 2000 is most likely because of the high IPv6 socket creation and connection times.

5 Conclusions

In this paper, we have conducted an empirical performance comparison of IPv4 and IPv6 protocol stack implementations of the most popular commodity operating systems including Windows 2000, Solaris and Linux and our experimental results demonstrate the impact that these IPv6 stacks are expected to have on end-user IPv6 applications. We found that:

- IPv6 (as well as IPv4) on Linux outperforms Windows 2000 and Solaris 8 IPv6 (and IPv4) implementations for all the metrics used. It is worthwhile pointing out that we obtained a minor degradation in throughput and round-trip latency performances for IPv6 compared to IPv4 on Windows 2000 and Solaris. However, we obtained improved (i.e. lower) round-trip latencies for IPv6 compared to IPv4 for Linux possibly because of a better coded IPv6.
- TCP/IPv4 and TCP/IPv6 socket creation times were sixteen times and nineteen times lower, respectively, on Linux compared to those on Windows 2000, and about four times lower (for TCP/IPv4 and TCP/IPv6) compared to Solaris.
- The web simulation results revealed a four-fold increase in performance (i.e. number of connections per second) for IPv6 on Solaris and Linux against IPv6 on Windows 2000. The fact that we obtained a degradation of 5%, 6% and 22% for Linux, Solaris and Windows 2000, respectively, demonstrates that IPv6-based web servers running Solaris or Linux will yield higher performance than those running Windows 2000.

6 Acknowledgments

This work was supported by grants (EDUD-7824-000145-US) from Sun Microsystems, Ixia Corporation and Microsoft. The authors are grateful to Ericsson (Denmark) for their Ericsson AXI 462 router and to IBM Corporation (Raleigh) for their IBM 2216 Nways Multiaccess Connector Model 400 router. They are also thankful to the anonymous reviewers for their suggestions and ideas. They thank C. Metz for his valuable feedback which helped to improve the originality, presentation and clarity of this paper. His suggestions to include Linux results led to an important contribution to this work. The authors express their gratitude to N. Goel for his ideas on the structuring of this paper and for his interesting remarks on the Linux results in particular. They would also like to thank F. Siddiqui for her help and suggestions on early versions of this paper and P. Wadehra for his help with the preparation of the graphs.

7 References

1 Information Sciences Institute, USC: 'Internet protocol'. RFC 791, IETF, September 1981
2 Deering, S., and Hinden, R.: 'Internet protocol, version 6 (IPv6) specification'. RFC 1883, Internet Engineering Task Force, December 1995
3 Goncalves, M., and Niles, K.: 'IPv6 networks' (McGraw–Hill, 1998)
4 Huitema, C.: 'IPv6: The new internet protocol' (Prentice Hall, 1997, 2nd edn.)
5 Zeadally, S., and Raicu, I.: 'Evaluating IPv6 on Windows and Solaris', *IEEE Internet Comput.*, 2003, **7**, (3)
6 Microsoft Corporation: 'Microsoft IPv6 technology preview for Windows 2000', December 2000, http://www.microsoft.com/windows2000/technologies/communications/ipv6/default.asp
7 Draves, R. *et al.*: 'Implementing IPv6 for Windows NT'. Proc. 2nd USENIX Windows NT Symposium, Seattle, WA, USA, August 1998
8 Anand, M.: 'Netperf3 TCP network performance on IPV6 using 2.4.17 kernel'. IBMLinux Technology Center, www-124.ibm.com/developerworks/opensource/linuxperf/netperf/results/may_02/netperf3_ipv6_2.4.17resutls.htm, August 2002
9 Ariga, S., Nagahashi, K., Minami, A., Esaki, H., and Murai, J.: 'Performance evaluation of data transmission using IPSec over IPv6 networks'. Proc. INET 2000, Japan, July 2000
10 KAME, http://www.kame.net
11 Ettikan, K.: 'IPv6 dual stack transition technique performance analysis: KAME on FreeBSD as the case'. Faculty of Information Technology, Multimedia University, Jalan Multimedia, October 2000
12 Jones, R.: Netperf, http://www.netperf.org/netperf/NetperfPage.html
13 Chuah, C., and Katz, R.: 'Characterizing packet audio streams from internet multimedia applications'. Proc. IEEE Int. Conf. on Communications (ICC 2002), New York, NY, USA, April 2002, pp. 1199–1203
14 Thompson, K., Miller, G., and Wilder, M.: 'Wide-area Internet traffic patterns and characteristics', *IEEE Netw.*, 1997, **11**, (6)
15 Ritchie, D.: 'A stream input-output system', *AT&T Bell Lab. Tech. J.*, 1984, **63**, (8), pp. 1897–1910
16 Jobst, M., and Feyrer H. 'IPv6 stack performance tests'. http://homepages/fh-regensburg.de/~jom30197
17 USAGI, http://www.linux-ipv6.org