Journal of Universal Computer Science, vol. 16, no. 21 (2010), 3299-3313
submitted: 20/10/09, accepted: 14/10/10, appeared: 1/12/10 © J.UCS

Impact of CPU-bound Processes on IP Forwarding of Linux and Windows XP

Khaled Salah
(Computer Engineering Department, Khalifa University of Science, Technology and Research (KUSTAR), Sharjah, UAE
khaled.salah@kustar.ac.ae)

Mohamed Hamawi
(Information and Computer Science Department, King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia
khaledhm@kfupm.edu.sa)

Abstract: These days, commodity-off-the-shelf (COTS) hardware and software are used to build high-end and powerful workstations and servers to be deployed in today's local area networks of private homes and small- to medium-sized business. Typically, these servers are multipurpose and shared - running networking functionalities involving IP packet forwarding in addition to other CPU intensive applications. In this paper we study and investigate the impact of running CPU-bound applications on the performance of IP packet forwarding. We measure and compare the impact and performance for the two operating systems of choice for home and small-business users, namely Linux and Windows XP. The performance is studied in terms of key performance metrics which include throughput, packet loss, round-trip delay, and CPU availability. For our measurements, we consider today's typical home network hosts of modern processors and Gigabit network cards. We also consider different configuration setups and utilize open-source tools to generate relatively high traffic rates. Our empirical results show that Linux exhibits superior performance over Windows XP in terms of IP forwarding performance. Results also show that, unlike Windows XP, the IP forwarding performance of Linux is not significantly impacted by running CPU-bound applications.

Keywords: Computer Networks, IP forwarding, Operating Systems, Linux, Windows, Network Performance
Categories: C.2, C.2.0, C.2.1, C.2.2, C.2.m

1 Introduction

These days, we witness a wide proliferation of Internet access to home networks. A home network is typically a wired or wireless local area network connecting multiple computing devices and home appliances, and with typically a single connection to the Internet via a residential gateway [Hwang and Tseng, 2005]. This enables home computing devices and appliances to connect simultaneously to the Internet, thereby allowing for remote surveillance, monitoring, and control. Computing devices typically include PC's, standalone web or multimedia servers, laptops, PDAs, smartphones, gaming devices, etc. Examples of widely deployed smart home appliances [Sandström et al., 2005]-[Smarthome, 2010] may include webcams, surveillance and fire sensors, devices for health monitoring and remote diagnosis,

intra-and inter-home communication systems, lighting and temperature control as well as plant watering and sprinkler control.

Two types of servers are commonly used in local area networks of private homes and small- to medium-sized business: (1) dedicated or standalone, and (2) shared. A dedicated or standalone server is a server that is dedicated to run one type of application with all of its software components installed on a single high-end PC. An example is the web server (such as the Apache HTTP server). Another good example commonly used in private home networks is a residential gateway which provides secure and simultaneous Internet access to home computing devices and appliances [Hwang and Tseng, 2005] through a dialup, cable, DSL, fiber, or WiMax modem. Some of today's modems provide a broadband speed from a few hundred Mbps to two Gbps and more. On the other hand, a shared server is multipurpose and typically runs multiple different applications in addition to performing networking functionalities. For example, a residential gateway can run other user applications and may also perform intrusion detection and firewall filtering.

Today's home networks are typically Gigabit Ethernet local area networks with servers, client machines, and switches connected at a speed of one and ten Gbps. Most modern PCs these days get shipped with Gigabit Ethernet NICs (Network Interface Cards), and the cost of these cards is relatively low. With such high-speed networks, the network performance bottleneck has become the end servers and workstations, particularly the ability to process network packets at Gigabit speed [Salah and El-Badawi, 2003], [Prytz and Johannessen, 2005]. Therefore the network performance of both dedicated and shared servers is critical. Any performance degradation can affect hosts connected to the private home network and can be noticed by both home and external users.

In order to be cost effective, commodity-off-the-shelf (COTS) hardware and software are used to build high-end and powerful workstations and servers to be deployed in today’s local area networks of private homes and small- to medium-sized business. For a home owner, a modern PC is typically the choice. This can be a PC with an Intel or AMD 3.2 GHz processor and 512 MB RAM. The preferred choice of the OS for a home owner these days is either Linux or Windows XP. Both OSes are the most widely used and readily available. The newly Microsoft operating system Windows Vista is having difficulties being adopted by the vast majority of users, as it requires more CPU power and hardware resources. As for COTS software, common utilities may come as part of the OS installation (such as Netfilter, DNS, DHCP, routing and NAT), or are typically open-source applications that have to be installed separately to run on the top of the OS such as [Apache, 2010] and [Snort, 2010].

A common problem (for a typical home owner or mid-size enterprise network engineer or administrator) is determining the most appropriate OS that gives the best networking performance while being able to run other user applications, some of which are typically classified as CPU-bound applications. CPU-bound applications are computationally intensive and require little or no I/O. As a practical example, a home user can setup a high-end machine that performs networking functionalities of forwarding and processing IP packets. Some examples of these machines may include PC-based routers, firewalls, intrusion detection and prevention (or commonly known as IDS or IPS) servers, Domain Name Service (or DNS) servers, IP address assignment (or DHCP) servers, and Network Address Translation (or NAT) servers.

In addition, the user may choose to utilize further this high-end machine by running other user applications such as multimedia and gaming.

There exists a handful of research articles in the literature on comparing application and network performance under Windows and Linux. In [Prytz and Johannessen, 2005], the performance of UDP stack on Windows XP and Linux was measured and compared. In [Salah and Hamawi, 2009], the performance in terms of packet forwarding of Linux, Windows Server and Windows XP were compared. User and kernel-level packet forwarding were measured. [Newman and Bush, 1999] presented a comparison of the performance of CPU-bound applications running on Linux and Windows 95/98/NT. Network applications were not considered. [Alfonsi and Muttoni, 2004] -[Kavas and Feitelson, 2001] examined the performance of running parallel applications in a cluster computing environment built from Linux and Windows NT machines. [Zeadally et al., 2004a] measured and compared the performance of different network application programming interfaces (APIs) under different OSes including Unix, Linux, Solaris, and Windows. The authors of [Zeadally et al., 2004b], [Mohamed et al., 2006] studied the performance of using IPv6 protocol stack over IPv4 of Windows and Linux-based hosts. The authors concluded that in general network applications running under Linux machines with IPv6 protocol stack outperform their respective Windows counterparts.

In this paper we focus primarily on gauging and assessing the impact of running user applications on the network performance of the two OSes of choice: Windows XP and Linux. Specifically, we present first an experimental comparative performance and analysis of networking performance in terms of IP packet forwarding, and then measure and study the impact of running CPU-bound applications on such performance. The networking performance of IP forwarding is studied in terms of throughput, packet loss, round-trip delay, and CPU availability. Our evaluation methodology is *general* and is a sufficient, standard, and popular evaluation methodology to assess network performance of servers, gateways, routers, and switches [Salim and Olsson, 2001] - [Brander and McQuaid, 1999]. In our experiments, we used open-source generators to generate relatively high traffic rates of up to 700 Kpps (packets per seconds).

The rest of the paper is organized as follows. Section II presents our evaluation methodology to assess and compare the performance of Linux and Windows XP. The section describes experimental setup, hardware and software configuration, traffic generation, and experimental tools. Section III reports and compares network performance measurements of Linux and Windows XP. In particular, the section compares the performance results for IP forwarding, and the impact of running user applications on IP forwarding. Finally, Section IV concludes the study and identifies future work.

2 Evaluation Methodology

In this section we describe our evaluation methodology to assess the network performance of both Windows XP and Linux. We describe briefly experimental setup, hardware and software configuration, traffic generation, and tools. More details about configuration and setup can be found in [Salah and Hamawi, 2009]. We measure the packet-forwarding performance of Linux and Windows XP machines, and also

measure the impact of running user applications on the performance of a host's packet forwarding. Packet-forwarding measurement is a standard evaluation methodology used in [Salim and Olsson, 2001] - [Brander and McQuaid, 1999] to assess the network performance of network devices such as servers, gateways, routers, and switches.

2.1 Experimental Setup

Figure 1 illustrates the basic test bed setup for packet forwarding. The experiment comprises two machines of a sender and a forwarder. The basic idea is to push the forwarder to its knees by having the sender generate high traffic and then measure the performance exhibited by the forwarder in its ability to forward packets back to the sender. The sender is a powerful dual-processor Linux machine that generates high traffic and the forwarder is the host under test which is a typical Linux or Windows XP machine used in home networks. Both machines have two physical NICs connected with 1 Gbps Ethernet crossover cables, as shown in the figure. Received packets on network interface eth1 are forwarded to network interface eth0.

The generator is a DELL PowerEdge 1800 machine equipped with two Intel Xeon processors running at 3.6 GHz with 4 GB RAM. It has an embedded Intel 82541GI Gigabit Ethernet NIC (which is eth0) running with the e1000 driver. The second interface eth1 is a 3COM Broadcom NetXtreme Gigabit Ethernet card with BCM5752 controller. The generator is installed with Fedora Core 5 running Linux 2.6.16. The forwarder is an HP Compaq DC7600 PC server equipped with Intel Pentium 4 processor running at 3.2 GHz with 512 MB RAM. It has two network interfaces (eth0 and eth1), and each is a 3COM Broadcom NetXtreme Gigabit Ethernet card with BCM5752 controller. The two versions of operating system that will be tested on the forwarder are Windows XP Professional and Linux 2.6.16 of Fedora Core 5.

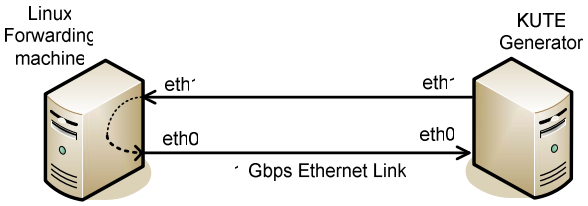

Figure 1: Test bed setup

2.2 KUTE Traffic Generator and Performance Metrics

To generate traffic from the sender machine, we used the open-source KUTE 1.4 traffic generator [Zander et al., 2005]. KUTE is a kernel-level generator capable of generating high UDP traffic rates. There are other two open-source and widely used traffic generators; namely, D-ITG and pktgen [Emma et al., 2010], [Olsson, 2005]. The pktgen is a kernel-level packet generator, while D-ITG is an application-level. Unlike D-ITG, KUTE can generate and receive UDP packets at much higher traffic rates. We were able to generate and receive packets at a rate of 700 Kpps with no

packet loss when directly connecting both eth1 and eth0 of the sender's machine with a Gigabit crossover cable. However, with D-ITG, the maximum generated rate with no packet loss was 235 Kpps. Also unlike pktgen, we had determined experimentally that KUTE has the ability to generate more accurate packet rates with finer granularity. It is worth mentioning that there are other open-source performance tools and benchmarks listed in [CAIDA, 2010] - [ITG, 2010]. We experimented with all of these traffic generation tools, and we found out that most of these tools and benchmarks do not generate high traffic rates as that of KUTE and pktgen.

KUTE has two kernel modules: namely, *kute_snd* and *kute_rcv* to send and receive packets, respectively. Statistics calculation of received packets is done by *kute_rcv*. Linux kernel 2.6.16 was patched to support *KUTE* packet reception in fast mode whereby *kute_rcv* receives packets immediately after the packets have been received by the NIC and before calling the kernel's default IP handler *ip_rcv*. The *kute_rcv* discards packets after gathering statistics. From these statistics, the average generated rate, throughput and packet loss can be calculated. However, KUTE does not measure RTT delay or the latency of sent and received packets, which is a key performance metric for our analysis and comparison. Therefore, we had to modify KUTE code to incorporate such an important measurement feature. To measure the average delay per packet, we timestamp the sent packet with the system clock at the sender machine. We used the kernel's *do_gettimeofday()* function which returns time in microsecond precision. When the packet is received, we calculate the packet RTT delay which is the difference between the current system clock and the timestamp set in the received packet. Whenever a new packet arrives, its delay is added to the total delay. To get the average delay per packet, this total delay is divided by the total number of received packets. It is important to re-calculate the UDP checksum after timestamping the packet to be sent, as the packet payload gets changed with a unique timestamp for each sent packet.

For our performance measurements and comparison, we considered the followings key performance metrics: average throughput, packet loss, RTT delay, and CPU availability. Average throughput measures how many packets per second the forwarding machine can sustain. Packet loss measures the percentage of packets dropped by the forwarding machine. RTT delay is the average delay per packet, as described earlier, and it takes into account the forwarding delay at the forwarding machine as well as other delays which include transmission, processing, queueing and propagation delays. CPU availability is measured at the forwarding machine and it represents the percentage of the available CPU time for user applications. In addition to measuring RTT delay of the forwarded UDP packets by KUTE, we use *ping* utility to measure the round-trip time of ICMP packets. It is to be noted that the performance metrics of throughput, packet loss, and RTT delay can also be classified in the literature as network performance metrics when describing the performance of link, routers, and other network elements. However, in this paper we use these parameters to reflect the performance of Linux and Windows XP operating systems when running both networking and computation applications.

We used KUTE to generate a traffic flow for a duration of 30 seconds with a specific sending rate. At the end of the flow, KUTE would report the total packets sent, received, and the average delay per packet. The average sending rate was recorded by calculating the total packets sent by KUTE in 30 seconds. The average

throughput was recorded by calculating total packets received by KUTE in 30 seconds. Packet loss was calculated by the following simple formula: ((the total packets sent by KUTE) – (the total packets received by KUTE)) / (the total packets sent by KUTE). Our final experimental results, shown in Section III, were the average of five experimental trials. Each 30-second flow generation would constitute one experimental trial.

The CPU availability at the forwarding machine was measured using the "*sar*" Linux utility. Specifically, we issued the shell command "sar -P ALL 2 3" at the forwarding machine a few seconds after starting the generation of the traffic flow. We issued the command after few seconds to ensure the desired traffic flow has stabilized enough and reached somewhat a steady state. This shell command computes the average of CPU idleness for all available CPUs. The average is the result of three readings which are taken two seconds apart. For Windows, we wrote a special VB Script to measure the CPU availability. The script uses API calls of the WMI (Windows Management Instrumentation) to read counters and collect system statistics. Specifically, the class *Win32_PerfFormattedData_PerfOS_Processor*, which is one of the many classes provided with WMI, was used to get the CPU usage statistics. It gets the CPU usage percentage via the member *PercentProcessorTime*. Similar to Linux measurement, we compute the average of three readings taken two seconds apart.

3 Performance Measurements

In this section we report and compare the performance of the forwarding machine under Linux and Windows XP for primarily three cases. In particular, we first compare performance measurements for kernel-level forwarding (or IP forwarding), and then examine the impact of running other user applications on IP forwarding performance. IP forwarding measurement is appropriate for gauging the performance of standalone servers such as a router or residential gateway. Studying the impact of user applications on IP forwarding is highly useful and beneficial in gauging the performance of shared servers, where for example user applications run on PC-based routers or gateways. The performance is compared and analyzed against different traffic load conditions generated by the generator machine.

3.1 IP Forwarding

Performance measurements of IP forwarding under Linux and Windows XP for throughput and packet loss are depicted in Figure 2. Figure 2(a) shows the throughput. Figure 2(b) shows the corresponding packet loss. It is clear from the figure that Linux substantially outperforms Windows XP. The figure shows that the maximum forwarding capacity for Linux and Windows XP are 350 Kpps and 75 Kpps, respectively. In addition it is important to notice that as incoming traffic load increases, Linux is able to sustain its performance at steady forwarding rate with no noticeable performance degradation in throughput. On the other hand, we notice that Windows XP throughput reaches a peak throughput at an incoming rate of 100 Kpps and then falls gradually with an increased incoming rate between 100 Kpps and 200 Kpps, before it sustains its forwarding rate at 75 Kpps.

The corresponding average round-trip delay per packet due to IP forwarding is shown in Figure 3(a). Figure 3(b) is a zoom-in version of Figure 3(a) to depict clearly the delay curve exhibited by Linux. It is obvious from these two figures that Linux has a substantial performance advantage over Windows XP, with a peak performance gain ratio of around 1:225. From Figure 3(b) it is shown that there is a sudden increase of delay around the area corresponding to severe packet loss of Figure 2(b). That is, in the case of Linux the sudden increase in delay is around 300 Kpps; and in the case of Windows XP it is around 50 Kpps. As we increase the incoming traffic rate, the delay for both systems reaches a peak value and then flattens off. This is the expected behavior of a typical queuing system. The maximum value is determined by the size of the queue and the mean forwarding capacity of the host under test.

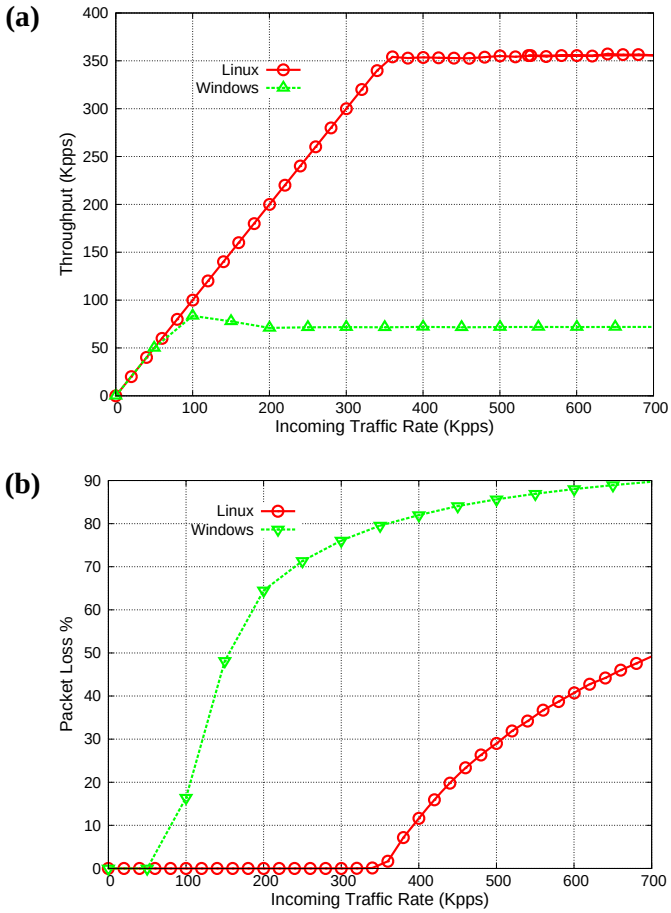

Figure 2: Throughput and corresponding packet loss of IP forwarding

Figure 4(a) plots and compares the curves of the corresponding CPU availability measured at the forwarding machine. There is an important observation that can be made from the curves shown in the figure. It is clear that Linux gives more CPU

availability at a region of an incoming traffic rate below 350 Kpps. However beyond a rate of 350 Kpps, the CPU availability of Linux drops to zero, as Linux kernel networking subsystem starts consuming all CPU power for packet forwarding. On the other hand, Windows always manages to leave some CPU power for user applications, as it is shown in the zoom-in version of Figure 4(a). It is clear from Figure 4(a) that Windows kernel starts consuming more CPU power with increased traffic rate.

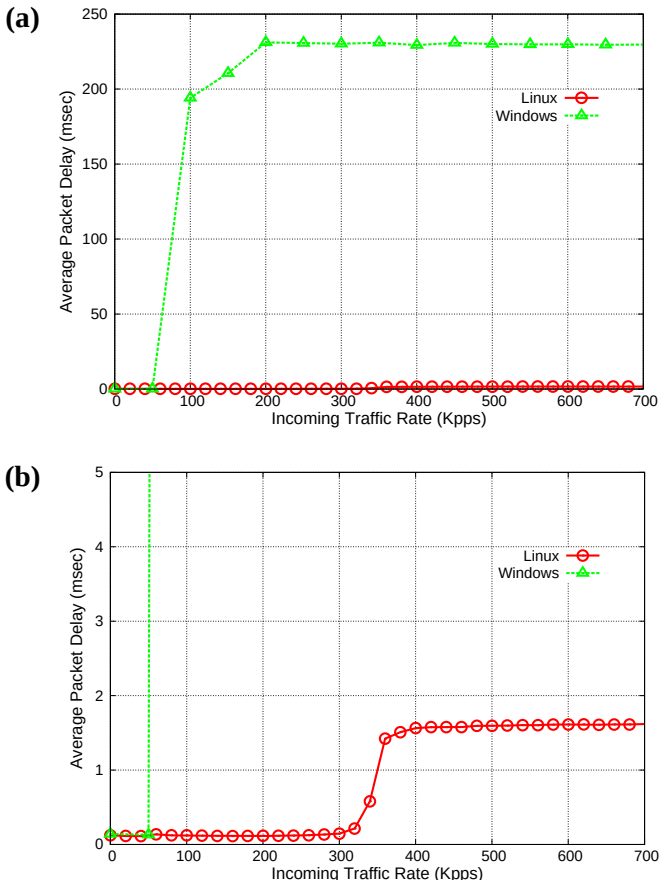

Figure 3: (a) Average round trip delay; (b) zoom-in delay

When observing closely Figure 4(a), we notice there is a noticeable fluctuation in the Linux curves for CPU availability. In particular, at an incoming rate of 50 Kpps, there is a short fall of CPU availability. Also there is similar behavior at around and shortly after an incoming rate of 200 Kpps, but conversely with a short rise of CPU availability. Such a fluctuation is primarily due to the interrupt overhead and handling. Interrupt overhead and handling always have precedence over other kernel and user tasks. In order to offer sound interpretations of such behavior, we measure and plot the corresponding interrupt rate for both receiving and transmitting NICs of

the forwarding machine, as shown in Figure 4(b). For measuring interrupt rates, we used the "sar" Linux utility. As depicted in Figure 4(b) there is an increase of interrupt rate, specifically for the receiving NIC, at an incoming rate of 50 Kpps and shortly after 200 Kpps. Such increase of interrupt rate is the primary reason for stealing CPU cycles from user processes, resulting in less CPU availability.

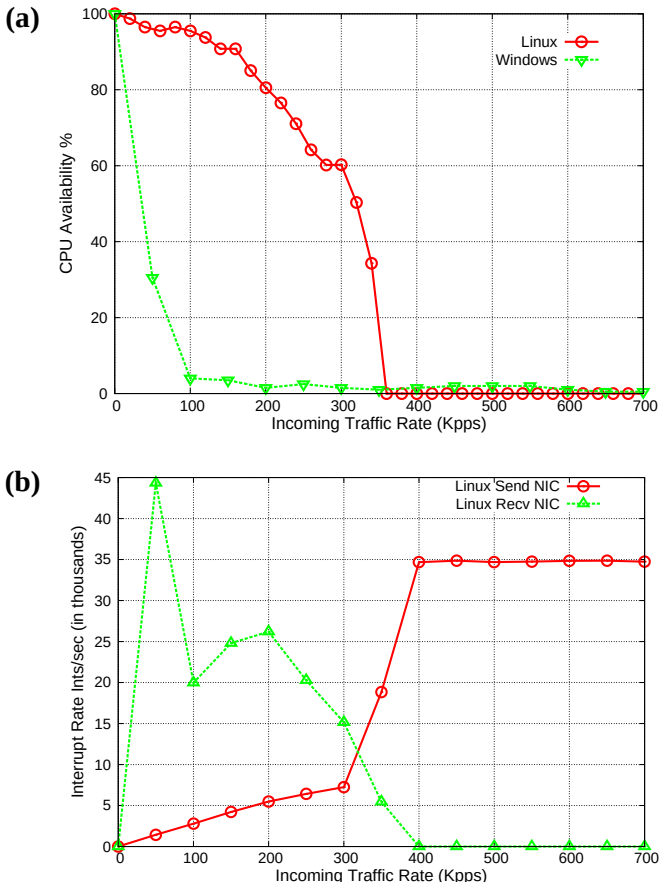

Figure 4: (a) CPU availability for user applications; (b) corresponding interrupt rate

Figure 4(b) shows that below a rate below 50 Kpps, the interrupt rate of the receiving NIC increases linearly in relation with the arrival rate. Shortly after that, the interrupt rate drops significantly, and then it starts increasing again. This is in line with the expected behavior. Below 50 Kpps, the interrupt rate is low and the forwarding machine is able to finish the interrupt handling and processing of packets before the next interrupt. The period of disabling and enabling RxInts (i.e. interrupt masking period) finishes before the occurrence of the next interrupt. As the incoming rate increases beyond 50 Kpps, multiple packet arrivals or interrupts occur within this masking period, thus showing a significant drop in interrupt rate shortly after 50

Kpps. After 60 Kpps, the interrupt rate starts increasing again but very slowly with respect to the arrival rate. Interrupt masking still occurs; however, the masking period is not stretching considerably with heavier traffic load. The CPU still has the power to handle and process packets relatively quickly. For the default NAPI budget of 300, at around 250 Kpps, the interrupt rate gradually drops to zero. This is due to awakening of *ksoftirqd* and switching to polling, whereby interrupts of the receiving NIC never get re-enabled as packets do not get exhausted in one polling period [30]. The *ksoftirqd* is a lower priority kernel thread and is used to prevent starvation of user processes. The curve of the interrupt rate of the transmitting NIC is straightforward, and reflects the actual throughput of the forwarding machine of Figure 2(a). The interrupt rate of transmitting NIC flattens off quickly when reaching the forwarding capacity. It is worth noting that there is also fluctuation in the curves of the CPU availability under Windows XP in Figure 4. Such fluctuation is most likely due to interrupt handling mechanism of Windows XP. The interrupt rate for Windows XP is not shown because we were not able to find a proper tool or system utility to measure it.

3.2 Impact of User Applications on IP Forwarding

In this section we examine and evaluate the impact of running user applications on IP forwarding. This is useful in gauging the performance of shared servers, where for example user applications run on PC-based routers or gateways. We examine such an impact under both Linux and Windows XP by measuring the performance of IP forwarding in terms of throughput, packet loss, and delay while running a CPU-bound application called Simplex. Simplex is a commonly used nonlinear numerical method for optimizing multi-dimensional unconstrained problems belonging to search algorithms. In particular, we used a modified version of the downhill simplex method code [Kaczmarczyk, 1999]. This program is a computationally heavy application with no disk or network I/O operations. The execution time is measured in microseconds.

For these measurements, we let Simplex execute for enough time so that it spans over the duration of the generated flow. We made sure that the flow started and finished while Simplex was still running. Specifically, the flow was generated for 30 seconds, and the Simplex program was run for around 35 seconds. Simplex was set to run two seconds prior to the start of the flow.

Figure 5 plots and compares the performance of IP forwarding with and without running Simplex. The performance is compared in terms of throughput, packet loss, and delay as depicted in Figures 5(a), (b), and (c), respectively. All of these figures clearly demonstrate that IP forwarding is not affected by running user applications. This is because Linux NAPI with the default budget of 300 gives more precedence to the underlying kernel's networking subsystem than user applications.

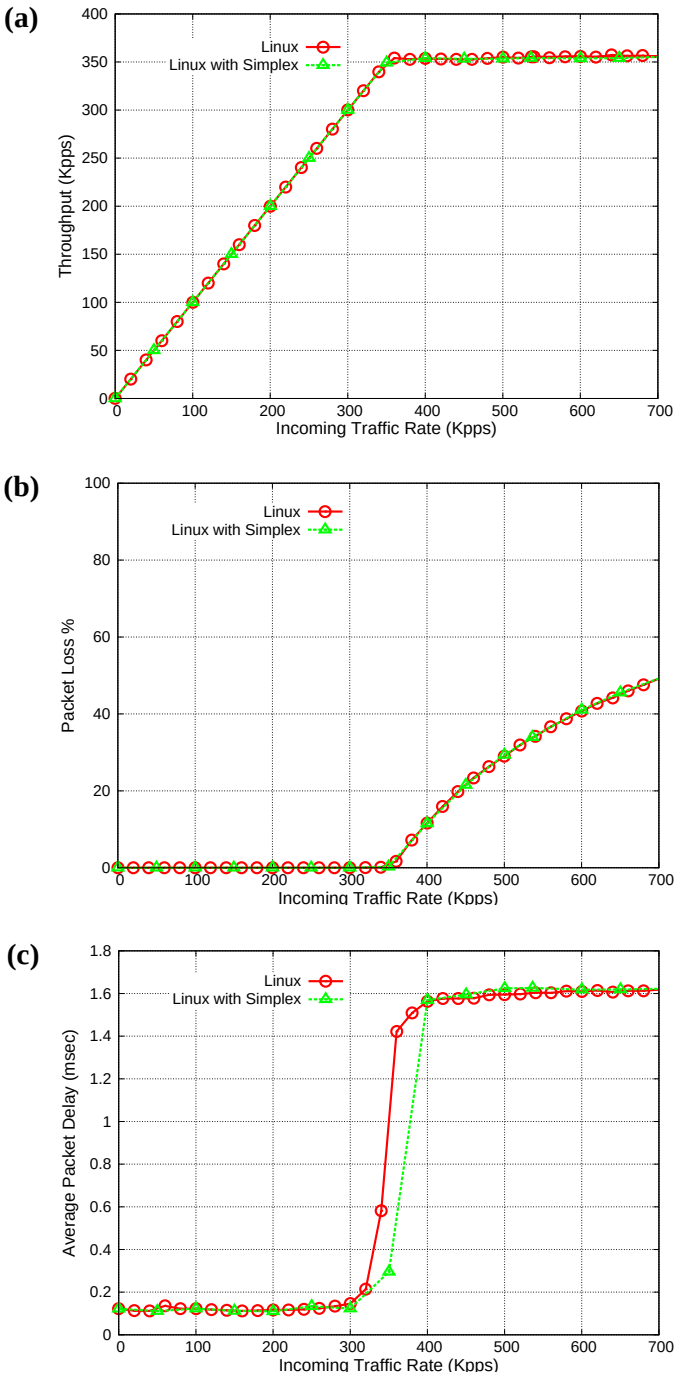

Figure 5: Impact of running Simplex on IP forwarding under Linux

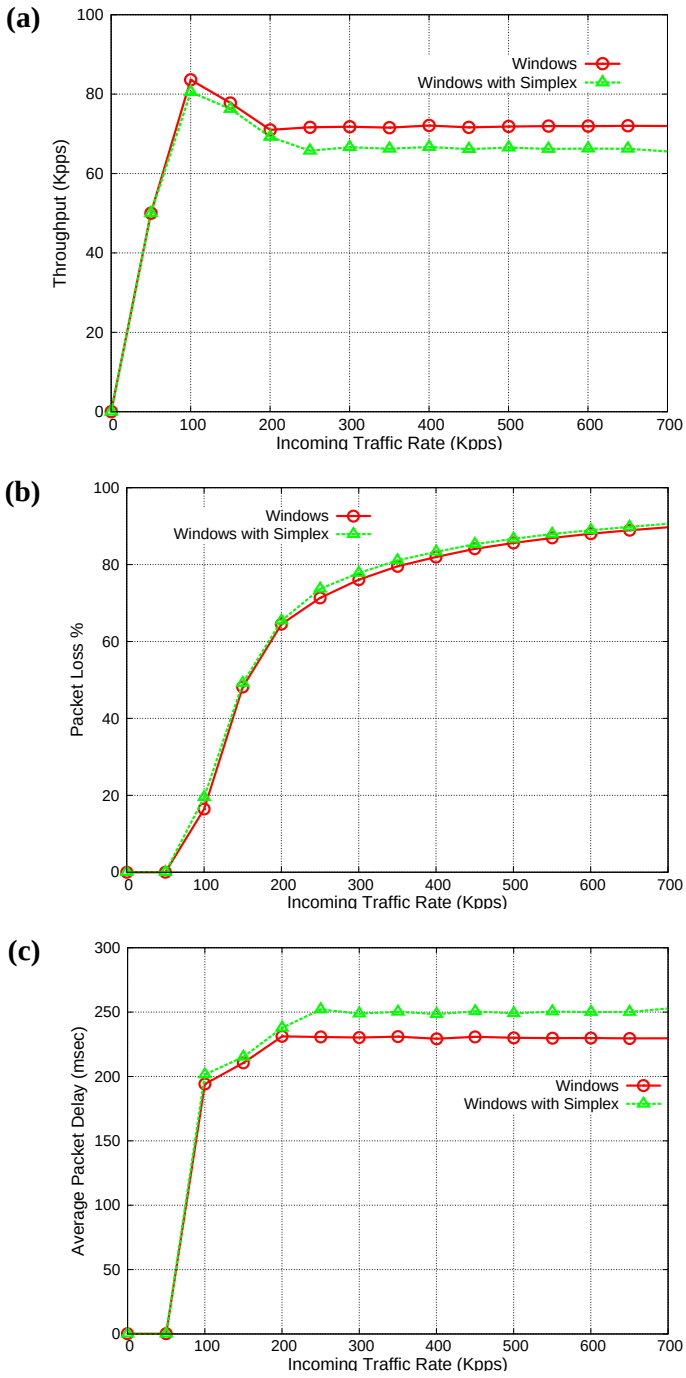

Figure 6: Impact of running Simplex on IP forwarding under Windows XP

Figure 6 depicts the performance of Windows XP packet forwarding while running the Simplex user application. In contrast to the impact on Linux IP forwarding performance where no or little impact was shown, Figure 6 demonstrates that running user applications evidently has a negative impact on Windows XP forwarding performance, particularly in moderate and heavy load regions. Figures 6(a) and (b) show that the negative impact on throughput and packet loss is around 5%; while the negative impact is around 8.5% for the delay as shown in Figure 6(c). It is clear that Windows XP steals and allocates more CPU processing power than Linux from its kernel's networking subsystem to user applications, and thus compromising its own IP forwarding capacity.

4 Concluding Remarks

In this paper we examined and compared experimentally the performance of Linux and Windows XP when used in private homes or small- to medium-sized business network environments. In particular we measured the packet-forwarding performance of a typical PC when deployed as a standalone or shared server. Packet-forwarding measurement is a standard and popular benchmark and evaluation methodology to assess network performance of network elements such as servers, gateways, routers, and switches. For a standalone server, the performance was measured for a number of key performance metrics which include throughput, packet loss, latency, and CPU availability. For a shared server, we measured and analyzed the impact of running user applications on the performance of IP forwarding.

Our empirical results show that Linux substantially outperforms Windows XP under almost all traffic load conditions. In addition, the networking performance of Windows XP is degraded when running other user application, with no obvious degradation in the case of Linux, and thereby making Linux the preferred platform of choice for standalone and shared networking servers. In a future study, we plan to examine and measure network performance of Windows XP and Linux on PCs with multi-core multiprocessor architecture.

Acknowledgements

We acknowledge the support of KUSTAR and KFUPM for the completion of this work. We are also very thankful to the authors of the traffic generators of KUTE and pktgen, Sebastian Zander and Robert Olsson, for assistance in getting their tools up and running.

References

[About, 2010] About.com, "TCP/IP Network Performance Benchmarks and Tools",Available at http://compnetworking.about.com/d/networkperformance/TCPIP_Network_Performance_Benchmarks_and_Tools.htm, Last visit: October 2010

[Alfonsi and Muttoni, 2004] Alfonsi, G., Muttoni, L.: "Performance Evaluation of a Windows NT based PC cluster for High Performance Computing," Journal of Systems Architecture, Elsevier Science, Vol. 50, No. 6, (2004), pp. 345-359

[Apache, 2010] The Apache Software Foundation, "Apache HTTP Server", http://www.apache.org/, Last visit: October 2010

[Bovet and Cesati, 2005] Bovet, D., Cesati, M.: “Understanding the Linux Kernel,” O’Riley Press, 3rd Edition, November 2005.

[Brander and McQuaid, 1999] Brander, S., McQuail, J.: "RFC2544 – Benchmarking Methodology for Network Interconnect Devices," March 1999.

[CAIDA, 2010] CAIDA.org, "Performance Measurement Tools Taxonomy," http://www.caida.org/tools/taxonomy/performance.xml, Last visit: October 2010

[Emma et al., 2010] Emma, D., Pescape, A., Ventre, G.: “D-ITG, Distributed Internet Traffic Generator”, Available from http://www.grid.unina.it/software/ITG, Last visit: October 2010

[Hwang and Tseng, 2005] Hwang, W.S., Tseng, P.C.: "A QoS-aware Residential Gateway with Bandwidth Management," IEEE Transactions on Consumer Electronics, Vol. 51, No. 3, August 2005, pp. 840-848.

[ITG, 2010] ITG: "Other Internet Traffic Generators," Available at http://www.grid.unina.it/software/ITG/link.php, Last visit: October 2010

[Kaczmarczyk, 1999] Kaczmarczyk, G.: “Downhill Simplex Method for Many (~20) Dimensions”, Available at http://paula.univ.gda.pl/~dokgrk/simplex.html, last modification: 1999.

[Kavas and Feitelson, 2001] Kavas, A., Feitelson, D.G.: "Comparing Windows NT, Linux, and QNX as the Basis for Cluster Systems," Journal of Concurrency and Computation" Practice & Experience, Vol. 13, No. 15, December 2001, pp. 1303-1332.

[Lancaster and Taked, 1999] Lancaster, D., Taked, K.: “Comparative performance of a commodity Alpha cluster running Linux and Windows NT”. In proceedings of the IEEE Workshop on Cluster Computing, May 1999.

[Meyer and Rakotonirainy, 2003] Meyer, S., Rakotonirainy, A.: "A survey of research on context-aware homes," In Proceedings of the Australasian information Security Workshop Conference on ACSW Frontiers 2003, Adelaide, Australia, pp. 159-168.

[Mohamed et al., 2006] Mohamed, S., Buhari, M., Saleem, H.: "Performance Comparison of Packet Transmission over IPv6 Network on Different Platforms," IEE Proceedings – Communications, Vol. 153, No. 3, June 2006, pp. 425-433

[Morris et al., 2000] Morris, R., Kohler, E., Jannotti, J., Kaashoek, M.: “The Click Modular Router,” *ACM Transactions on Computer Systems*, vol. 8, no. 3, August 2000, pp. 263-297

[Newman and Bush, 1999] Newman, T., Bush, J.: "Performance Comparison," Linux Journal, Vol. 1999, No. 7, November 1999.

[Olsson, 2005] Olsson, R.: “pktgen the Linux Packet Generator,” Proceedings of Linux Symposium, Ottawa, Canada, 2005.

[Prytz and Johannessen, 2005] Prytz, G., Johannessen, S.: "Real-Time Performance Measurements using UDP on Windows and Linux", The 10th IEEE Conference on Emerging Technologies and Factory Automation, 2005, Catania, Italy, September 19-22, 2005, pp. 925-932

[Salah and El-Badawi, 2003] Salah, K., El-Badawi, K.: " Evaluating System Performance in Gigabit Networks ", The 28th IEEE Local Computer Networks (LCN), Bonn/Königswinter, Germany, October 20-24, 2003, pp. 498-505

[Salah and Hamawi, 2009] Salah, K., Hamawi, M.: "Comparative Packet Forwarding Measurement of Three Popular Operating Systems," International Journal of Network and Computer Applications, Elevier Science, Vol. 32, No. 5, September 2009, pp. 1039-1048

[Salim and Olsson, 2001] Salim, J. H., Olsson, R.: "Beyond Softnet," Proceedings of the 5th Annual Linux Showcase and Conference, November 2001, pp 165-172

[Sandström et al., 2005] Sandström, G., Gustavsson, S., Lundberg, S., Keijer, U., Junestrand, S.: "Long-Term Viability of Smart Home Systems -- Business Modelling and Conceptual Requirements on Technology," Proceedings of. Home-Oriented Informatics and Telematics, (HOIT 2005), 2005, pp. 71–86.

[Smarthome, 2010] Smarthome.com, Available at http://www.smarthome.com, Last visit: October 2010

[Snort, 2010] Snort.org, "Snort - the De Facto Standard for Intrusion detection and Prevention," Available at http://www.snort.org, Last visit: October 2010

[Zander et al., 2005] Zander, S., Kennedy, D., Armitage, G.: "KUTE - A High Performance Kernel-based UDP Traffic Engine," Center for Advanced Internet Architectures (CAIA). Technical Report 050118A, January 2005.

[Zeadally et al., 2004a] Zeadally, S., Zhang, L., Zhu, Z., Lu, J.: "Network Application Programming Interfaces (APIs) Performance on Commodity Operating Systems," Journal of Information and Software Technology, Elsevier Science, Vol. 46, No.6, May 2004, pp. 397-402.

[Zeadally et at., 2004b] Zeadally, S., Wassem, R., Raicu, I.: "Comparison of End-System IPv6 Protocol Stacks," IEE Proceedings – Communications, Vol 151, No. 3, June 2004, pp. 238-242.