Journal of Network and Computer Applications 36 (2013) 452–465

Contents lists available at SciVerse ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Performance of IP-forwarding of Linux hosts with multiple network interfaces

K. Salah [a,*], M. Hamawi [b]

[a] Electrical and Computer Engineering Department, Khalifa University of Science, Technology and Research (KUSTAR), PO Box 573, Sharjah, United Arab Emirates
[b] Department of Information and Computer Science, King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia

ARTICLE INFO

Article history:
Received 16 January 2012
Received in revised form
10 April 2012
Accepted 27 April 2012
Available online 7 May 2012

Keywords:
Network performance
IP forwarding
Linux
SMP
Multicore architecture

ABSTRACT

This paper evaluates and compares the performance of IP-packet forwarding of a Linux host equipped with multiple network interface cards (NICs), namely two receiving NICs and one transmitting NIC. We consider a Linux host with SMP (Symmetric Multiprocessing) or multicore multiprocessor (MCMP) architecture. We measure IP forwarding by subjecting an MCMP Linux host to different traffic load conditions of up to 1 Gbps. We used the IXIA hardware traffic generator to generate traffic with fixed- and variable-size packets. At the Linux host, generated packets are forwarded/routed from the two receiving NICs to the transmitting NIC. We consider two NIC affinity modes: (I) both receiving NICs are affinitized (or bound) to two cores of the same processor while the transmitting NIC is affinitized to a core on a separate processor, and (II) the transmitting NIC and one receiving NIC are affinitized to two cores of the same processor while the second receving NIC is affinitized to a core on a separate processor. For each affinity mode, we measure the performance for three packet reception mechanisms: NAPI (New API) with a default budget of 300, NAPI with a budget of 2, and Disable and Enable interrupt handling. The performance is measured and compared in terms of various key performance metrics which include throughput, packet loss, round-trip delay, interrupt rates, and CPU availability.

1. Introduction

The majority of today's hardware design and motherboard architecture of commercial and PC-based network nodes and hosts involve the use of multiprocessors, with each processor having multiple cores. Such architecture is known as Multicore Multiprocessor (MCMP), and commonly known as Symmetric Multiprocessing (SMP) architecture where two or more identical processors connect to a single shared main memory. SMP operating systems, such as that of Linux, treat all processing elements (i.e., cores) equally. Some examples of commercial routers and switches that utilize MCMP/SMP architecture include Cisco 7500 (White et al., 2000) and Nortel Contivity 4500 VPN switch (Nortel Inc). Although special-purpose hardware is used to forward packets, many other tasks are still implemented in software and are being decomposed to run in parallel utilizing MCMP hardware. Some of these tasks include network address translation, encrypting virtual private network tunnels, and sophisticated packet queuing and scheduling.

More importantly, considerable research work and projects have been directed toward architecting and designing open-source and PC-based network nodes, applications, services, and OSes to be augmented to take advantage of the capabilities of today's COTS (commodity off-the-shelf) motherboard with MCMP hardware. Some of these popular research projects include the Click Modular Router Project (Morris et al., 2000) and its recent symmetric multiprocessing versions of SMPClick (Chen and Morris, 2001; Zebra, Inc; Guagga; Xorp, Org; Handley et al., 2003). The research work presented in Leroux (2001, Microsoft Inc (2004), Bolla and Bruschi (2006, 2008a) demonstrated the need for OS and PC-based network software to take advantage of MCMP hardware in order to support efficiently multi-gigabit network traffic. Experimental results reported in Olsson et al. (2008), Foong et al. (2005), Bolla and Bruschi (2008b) showed that free affinity (also known as free or dynamic affinity) which is performed by the Linux scheduler yields poor performance. Also, Bolla and Bruschi (2008b), Markatos and LeBlanc, Subramaniam and Eager (1994) reported a potential performance improvement by affinitizing kernel processes/threads to processors in an SMP architecture.

This paper focuses on studying and measuring the network performance of Linux hosts equipped with multiple network interfaces. We consider a Linux host with SMP (Symmetric Multiprocessing) or multicore multiprocessor (MCMP) architecture.

* Corresponding author. Tel.: +971567075690.
E-mail addresses: khaled.salah@kustar.ac.ae (K. Salah), hamawi@kfupm.edu.sa (M. Hamawi).

1084-8045/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2012.04.013

Specifically, we measure kernel- (or IP-) packet forwarding by subjecting an MCMP Linux host to different traffic load conditions of up to 1 Gbps. We use the IXIA generator/analyzer to generate packets to the Linux host where packets are then forwarded/routed from two receiving network interfaces (Rx NICs) to a different transmitting network interface (Tx NIC). As opposed to publically available software-based traffic generators such as KUTE (Zander et al., 2005) and pktgen (Olsson, 2005; Emma et al., 2004), IXIA is a powerful hardware-based traffic generator capable of generating and analyzing traffic at wire speed. In our experimental work, we consider traffic with fixed- and variable-size packets. Our experimental evaluation methodology is a standard and popular benchmark and evaluation method to assess the performance of network elements such as servers, gateways, routers, and switches. Examples of servers performing networking functionalities at the kernel level may include PC-based routers, firewalls, Domain Name Service (or DNS) servers, IP address assignment (or DHCP) servers, Network Intrusion Detection Systems (NIDS) as that of Snort, and Network Address Translation (or NAT) servers (Siris and Stavrakis, 2007; Huang et al., 2009; Mohamed and Al-Jaroodi, 2010; Salah and Kahtani, 2010; Salah and Hamawi, 2009).

Unlike prior simulation and empirical studies reported in the literature on packet forwarding performance of MCMP/SMP systems (Chen and Morris, 2001; Leroux, 2001; Bolla and Bruschi, 2006, 2008a, 2008b; Olsson et al., 2008; Foong et al., 2005; Markatos and LeBlanc; Subramaniam and Eager, 1994), our work presented in this paper is unique and has the following key contributions. First, we measure and compare the performance of today's MCMP Linux hosts with multiple network interfaces. Second, we consider two types of static NIC affinity modes and consider three packet reception schemes, namely NAPI (New API) with a default budget of 300, NAPI with a budget of two, and Disable and Enable interrupt handling. We refer to these mechanisms as NAPI(300), NAPI(2), and DE. Third, we identify and discuss hardware and software architectural bottlenecks that can play a major role in impacting the performance of IP forwarding. This discussion gives insights into system behavior and offers sound interpretations of empirical results. Fourth, we employ an evaluation methodology for measuring IP forwarding performance by using the IXIA traffic generator and analyzer. Fifth, we measure and compare the performance for a number metrics which include throughput, packet loss, delay, interrupt rates, and CPU availability. More importantly, we conduct all these measures with regards to traffic of fixed- and variable-size packets. Sixth, the paper offers useful information on experimental setup, system configurations, and Linux reception and forwarding mechanisms. Seventh, our recommendations for improving IP forwarding of MCMP Linux hosts with multiple NICs are unique, and in fact can be orthogonal to other improvements reported in the literature, which may include affinitization of certain tasks, processes or threads. To the best of our knowledge, no study or work in the literature addresses the performance of packet reception mechanisms under the stated NIC affinity modes.

Unlike our prior work presented in Salah et al. (2010) which considered only one receiving NIC and one transmitting NIC, this paper measures and compares the performance considering three network interface cards: two receiving and one transmitting. In Salah et al. (2010), we have considered only a system with two network cards with one receiving NIC and one transimitting NIC. However in this contribution, we have considered a more practical and realistic scenario for PC-based routers or network servers, in which more than one receiving NICs are used.

The rest of the paper is organized as follows. Section 2 describes IP forwarding in Linux with the different packet reception mechanisms. The section also describes a typical MCMP/SMP architecture of a Linux host, and it identifies hardware and software architectural bottlenecks that can significantly impact the performance of IP forwarding. Section 3 presents our evaluation methodology to measure and compare the performance of IP forwarding. The section also describes experimental setup, traffic generator/analyzer and important system configuration details. Section 4 reports and compares network performance measurements of the two affinity modes under the three reception mechanisms of NAPI(300), NAPI(2), and DE. The performance is reported for incoming traffic containing fixed- and variable-size packets. Finally, Section 5 concludes the study and identifies future work.

2. Linux IP forwarding

Forwarding of network packets at layer 3 by the Linux kernel starts by having the NIC DMA'ing incoming packets into the RxRing buffer located in the kernel memory. A hardware IRQ interrupt gets triggered after successfully DMA'ing each packet into the RxRing, notifying the kernel to start processing packets by Rx API reception mechanisms. Three possible Rx API mechanisms exist, namely NAPI(300), NAPI(2), and DE. A detailed description of these is discussed in Section 2.1. As shown in Fig. 1, the reception mechanism performs layer 2 and layer 3 delivery operations for each packet up to the routing/forwarding decision, and then each packet is queued into the TxRing for transmission by the Tx NIC. The Tx API is responsible for moving TxRing packets into the NIC buffer for transmission and de-allocating all already transmitted packets from the TxRing. Tx API operation is carried out by SoftIRQ which is a high-priority kernel function. It is to be noted that Tx API operation is performed in a similar fashion for all reception mechanisms of NAPI(300), NAPI(2), and DE.

2.1. Reception mechanisms

In this section, we discuss three types of reception mechanisms, and we highlight their computational cost and behavior that impact the performance of IP forwarding.

NAPI(300) and NAPI(2). New API (NAPI) (Salim et al., 2001) is a packet reception mechanism which is implemented in Linux 2.6 to alleviate the problem of *receive livelock* (Salim et al., 2001) by a using combination of interrupts and polling mechanism. In *receive livelock*, CPU power is totally consumed by interrupt handling resulting from high interrupt rates of incoming packets, thereby starving other kernel tasks and user applications. To alleviate this problem, NAPI disables the received interrupt (RxInt) of the network device upon the arrival of a packet to stop the generation of further interrupts and then raises a SoftIRQ to schedule polling. SoftIRQ is a non-urgent (or deferrable) interruptible kernel event that gets handled by *__do_softirq* kernel function. *__do_softirq* gets invoked (if there are any pending softirqs) typically at the end of I/O and timer interrupts (Bovet and Cesati, 2005).

Within a single SoftIRQ invocation, up to budget B packets are processed for all network devices or interfaces and up to quota Q packets are processed per interface. This is done to ensure fairness of packet processing in the case of a host having multiple network interfaces. *poll_list* is used to process packets per network interface in a round-robin fashion. In Linux 2.6, the budget B and quota Q are set to 300 and 64, respectively. **NAPI(300)** is the default

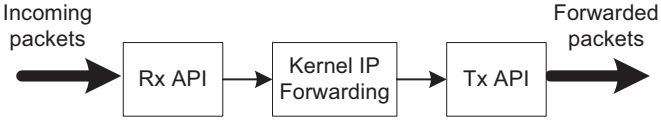

Fig. 1. Main software components of IP forwarding in Linux kernels.

NAPI scheme running with a default budget of 300. **NAPI(2)** runs with a modified budget value of 2. The value of the budget B plays a major role in controlling the sharing of the CPU bandwidth between networking subsystem and user-level processes. Small budget B values will prevent user-level processes from starving, especially under heavy network traffic load conditions. In other ways, small budget B values will limit the number of packets being processed by one SoftIRQ invocation.

DE. The idea of interrupt disable-enable (or DE) scheme (Mogul and Ramakrishnan, 1997; Salah et al., 2007) is to turn off (or disable) the received interrupts of incoming packets as long as there are packets in the DMA RxRing, i.e., the RxRing is not empty. When the RxRing is empty, the interrupts are turned on again (or re-enabled). This means that protocol processing of packets by the kernel is processed immediately l. It is worth noting that using DE, incoming packets get processed as fast as possible with interrupt-level priority. Unlike other interrupt-handling schemes, DE will minimize buffer overflow. Any incoming packets (while the interrupts are disabled) are DMA'd quietly to the protocol buffer without incurring any interrupt overhead. DE mechanism has been implemented in Linux 2.6, and details of implementation can be found in Salah and Qahtan (2009).

As shown in Fig. 1, after packet reception and IP processing of packets (which is done exhaustively at the interrupt level in the case of DE or via SoftIRQ of up to B packets), the packets are placed in the TxRing of the transmitting NIC. In both affinity modes under study, interrupts of TxRing are handled by a separate CPU core, as shown in Fig. 3. Upon the transmission of a packet from TxRing, an interrupt is triggered for polling the next packet from the TxRing and placing it for transmission in the NIC transmission buffer. Transmission of packets is also done by SoftIRQ. TxRing data structure can effectively be accessed by SoftIRQ code being run in parallel by multiple cores. One core is queueing a packet to be routed in TxRing, and the other core is polling a packet from TxRing to place it in the NIC transmission buffer. In order to access the shared structure of TxRing in a mutual exclusion fashion (i.e., only one SoftIRQ at a time), a spin-lock function with a lock (called "LLTX") is used. This serializes concurrent SoftIRQ accesses by different cores to the TxRing. In other words, when a SoftIRQ gains access to the TxRing, it temporarily stops the following SoftIRQs, which must wait for the LLTX lock to be released.

2.2. MCMP architecture

The IP-packet forwarding performance is affected by the MCMP architecture of the forwarding host. Fig. 2 shows a typical MCMP motherboard for a Dell PowerEdge 1900 Server. We will use this machine in our experimental testbed. The motherboard contains two identical processors, and each processor contains two cores. As shown, the Dell PowerEdge 1900 Server main board has dual-core Intel Xeon E5310 processors running at 1.6 GHz. Each processor has 4 MB of L2 cache, 4 MB of L3 cache, 1 GB of RAM, and an independent FSB (Front-Side Bus) to the chipset. The FSB runs at a speed of 1333 MHz, and it is capable of transferring up to 21 GB/s of aggregated throughput for both buses. The chipset is Intel 5000 P, which is the a bus arbiter that controls and arbitrates RAM and I/O access. The two receiving network cards (Rx NIC 1 and Rx NIC 2) and the transmitting card (Tx NIC) are connected to the chipset by the PCI Express (PCIe) shared bus. All of the three NICs are 3COM Broadcom NetXtreme Gigabit Ethernet cards with BCM5752 controller.

2.3. Performance bottlenecks

In this section we identify the major software and hardware architectural bottlenecks that impact the performance of IP forwarding. Identifying these bottlenecks gives insights on interpreting performance results (as will be demonstrated in Section 4), and therefore proper affinity mode and packet reception mechanisms can be selected. Additionally, identifying these bottlenecks would help in recommending possible future improvement to both hardware and software. We identify first two software architectural bottlenecks and then two hardware architectural bottlenecks:

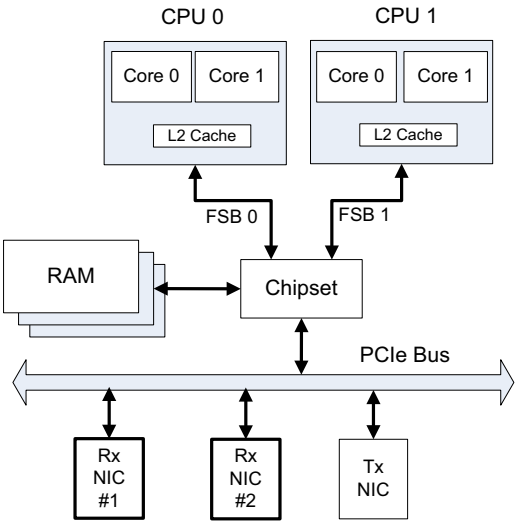

Fig. 2. MCMP architecture with multiple NICs.

(1) **Computational Cost and Behavior of Reception Mechanisms.** The computational cost and behavior of reception mechanisms can impact the performance negatively. It is crucial to note that DE incurs far less computational cost than NAPI for a number of reasons. First, DE starts the processing of packets immediately at interrupt level, whereas NAPI defers the processing of packets and it gets executed at a lower priority by using SoftIRQ. Second, all SoftIRQs are reentrant, i.e., they run with interrupts enabled and therefore can be preempted at any time to handle a new incoming interrupt, so that the handling of a SoftIRQ may stretch due to other interrupt handling activities in the system. Third, a SoftIRQ may also stretch considerably due to processing of other SoftIRQs, as *__do_softirq* handles not only SoftIRQ of receiving packets, but also five other softirq types which include soft timers, high and low-priority tasklets, transmission of packets, and SCSI handling. Fourth, NAPI incurs non-ignorable I/O write latencies to disable and enable Rx interrupts (Salim et al., 2001), and it also requires the enforcing of upper bounds for budget, quota, and handling time. In contrast, DE simply performs exhaustive processing of received packets.

(2) **"LLTX" Lock Contention.** It is clear from the discussion in Section 2.1 that serialization of parallel SoftIRQ accesses by different cores to shared TxRing structure plays a crucial role in impacting the performance. It is important also to note that under heavy traffic load conditions, DE and NAPI(300) causes more "LLTX" lock contention due to the longer access period resulting from the exhaustive nature of packet processing. By contrast, NAPI(2) limits the access period by having a smaller budget and thereby limiting the processing of packets to be transferred to the shared TxRing in one SoftIRQ invocation.

(3) **L2 Cache Utilization and Invalidation.** As shown in Fig. 2, data access to packet descriptors (to be manipulated and transferred from the RxRing to the TxRing) can be faster if the

L2 cache is utilized by the concurrent SoftIRQs being run by cores located on the same processor. Slow memory access occurs when the L2 cache is not utilized in the case when RxRing and TxRing are affinitized to separate cores located on separate processors. In addition, L2 cache invalidation can occur between separate processors whenever a modify access to the shared TxRing occurs. The reason for invalidation is primarily to maintain cache coherency or synchronization whereby each processor sees the same value for a data item in its local cache as the values in the other processor's caches. This state is transparent to software, but maintaining such a state can degrade the performance significantly. Whenever a processor's core accesses a packet descriptor in TxRing, all of the other processors caching it must invalidate their cache copies. If a series of a modify access to TxRing is done by different cores on separate processors, the performance is degraded significantly because CPU cycles are wasted on cache synchronization. This situation is also known as cache *pingponging* or *bouncing*.

(4) **FSB Bus Contention.** As shown in Fig. 2, more FSB contention occurs when two NICs are affinitized to two cores of the same processor, whereas no FSB contention occurs when one NIC is affinitized to only one core of a processor. In the latter, bus bandwidth *in theory* doubles to the chipset, resulting in a bandwidth boost between core and memory, and also between core and I/O including interrupt delivery.

2.4. NIC affinity modes

Based on the above software and hardware architectural bottlenecks, two possible static NIC affinity modes emerge: Affinity Mode I and Affinity Mode II. We focus on using static affinity as opposed to free (or dynamic) affinity. In free affinity, the Linux scheduler allocates computational tasks and processes to cores (logical CPUs) based on a set of rules to balance the processing load over all the available cores. Free affinity is not recommended for IP forwarding, and it yields poor performance as reported in Olsson et al. (2008), Foong et al. (2005), Bolla and Bruschi (2008), Salim et al. (2001). Our static affinity modes to be examined are illustrated in Fig. 3. As shown, Affinity Mode I affinitizes the two receiving NICs (namely Rx NIC 1 and Rx NIC 2) to two cores of the same processor while the transmitting NIC (Tx NIC) to a core on a separate processor. On the other hand, Affinity Mode II affinitizes one of the receiving NICS (Rx NIC 1) while the transmitting NIC (Tx NIC) to two cores of the same processor and the second receiving NIC (Rx NIC 2) to a core on a separate processor. It is to be noted that any other NIC affinitization to cores will be equivalent in terms of performance to any of these two affinity modes. This has been proved experimentally and it is in line with intuition.

3. Evaluation methodology

In this section we describe our evaluation methodology to assess the performance of IP forwarding using the two affinity modes, considering the three packet reception mechanisms. We describe experimental setup, hardware configuration, traffic generator and analyzer, and system configuration details.

3.1. Experimental setup

The testbed setup for IP forwarding is illustrated in Fig. 4. The experiment comprises a IXIA 400 T hardware traffic generator and a Linux forwarding machine. The basic idea is to investigate the performance limits of the Linux networking subsystem by having the IXIA 400 T generate high traffic and then measuring the performance exhibited by the forwarder in its ability to forward packets back to IXIA 400 T analyzer. The Linux forwarding machine and IXIA 400 T are connected with 1 Gbps Ethernet cables, as shown in the figure. In our experiment, only two IXIA Gbps ports were available. Packets are generated from interface 1 (*if1*) of IXIA 400 T and received back on for analysis on interface 2 *(if2)*, as shown in Fig. 4.

Our forwarding machine is the Dell PowerEdge 1900 Server, described and shown in Fig. 2. The NICs are running with a loadable kernel module of the modified BCM5700 driver version 8.2.18 that implements the scheme of DE. The modified BCM5700 driver has the implementations of our proposed scheme of NAPI and DE schemes. The Linux machine runs with Fedora Core 9 Linux 2.6.35. We replaced Fedora kernel with vanilla Linux kernel 2.6.35. The vanilla Linux kernel is the kernel whose code is approved and released by Linus Torvalds, the creator of Linux. To minimize the impact of other system activities on performance and measurement, we booted up the Linux machine with run level 3, and we made sure that no services were running in the background. We also disabled Ethernet link flow control.

To generate and analyze traffic to and from the Linux forwarding machine, we used the IXIA T400 Traffic Generator/Analyzer (IXIA Inc.). IXIA T400 is a hardware traffic generator and analyzer which comes equipped with multiple Gigabit Ethernet ports. The IXIA analyzer measures delays with high precision and accuracy of microsecond granularity. The IXIA generator is capable of generating packets at full Gigabit speed, filling up the full 1 Gigabit pipe. This means around 1.4 Mpps (packets per second) can be generated using fixed-size packets of a minimum size of 64-byte. The IXIA generator has also the ability to generate traffic of variable-size packets according to predefined distribution. In our experiments, we will use IXIA to generate both fixed- and variable-size packets.

Since we had only one transmitting IXIA port available and two receiving NICs, we used the CC1200-P network tap

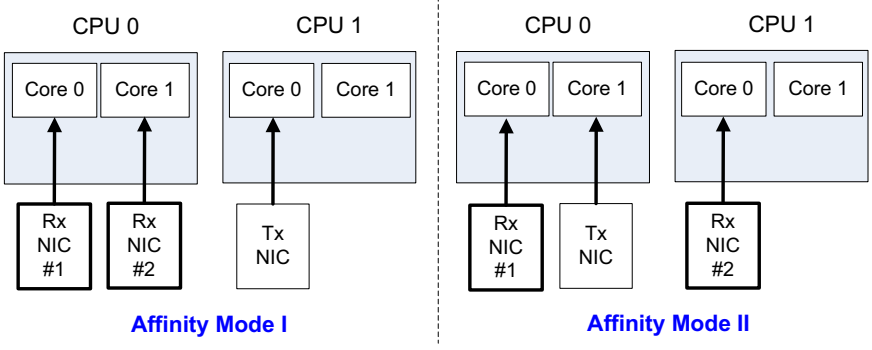

Fig. 3. Mappings of NICs to CPU cores.

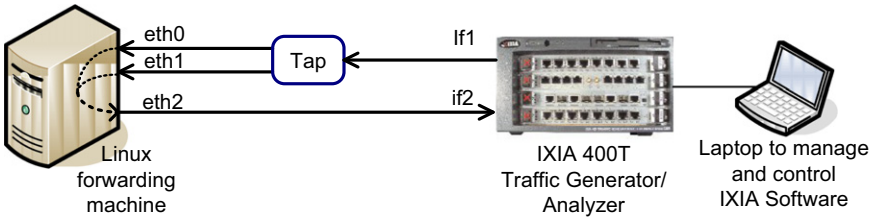

Fig. 4. Experimental setup using IXIA hardware generator.

(NetworkCritical Inc.) to send to the same gigabit traffic to both *eth0* and *eth1* network interfaces of the Linux machine. In order to properly route packets from these two different network interfaces, we setup IXIA software to generate two interleaving flows, with each traffic flow is destined to a specific MAC address. Effectively, the generated traffic which gets received by each interface is split equally. For example, with the maximum packet rate of 1.4 Mpps, the effective received incoming rate at both *eth0* and *eth1* will be around 700 Kpps. As demonstrated in Section 4.2, there was no need for two separate transmitting IXIA interfaces. One transmitting IXIA interface was adequate when used with the CC1200-P network tap to split it into two. The split traffic of 700 Kpps was more than enough as the receiving NICs were only able to handle a traffic of up to 500 Kpps.

IXIA 400 T is controlled and managed by IxExplorer management software (IXIA Inc.), which ran on a laptop as shown in Fig. 4. For all results reported and shown in this section, we performed five experimental trials, and the final results are the average of these five trials. For each trial, we recorded the results after the generation of a traffic flow with a specific rate for a sufficient duration of 30 s. Longer durations made negligible differences to the results.

3.2. System configuration details

In this section we list important configurations used in our experimental setup and evaluation. We specifically present useful details for setting NIC affinity, configuring packet reception for NAPI(300) and NAPI(2), setting routing/forwarding, disabling flow control, and minimizing system background services and activities.

Setting NIC Affinity. The affinitization of NIC to the specific core was done by using the kernel exposed system variable *smp_affinity* that is available for every IRQ number. This exposed system variable is accessible through/proc file system as "/proc/irq/〈IRQ number〉/smp_affinity". The IRQ numbers for the Rx and Tx NIC can be known by issuing the command "$cat/proc/interrupts". In our case, the Rx1,Rx2, and Tx NICs have IRQs of 16, 18, and 20, respectively. In Linux, each core is represented as a logical CPU. For our MCMP host, shown in Fig. 4, we have four logical CPUs, namely CPU0, CPU1, CPU2, and CPU3. To affinitize 〈IRQ number〉 to a certain CPU, *smp_affinity* has to be set with the proper bitmask for that CPU.

To set Affinity Mode I, we issue the following two commands:

```
$ echo 01〉/proc/irq/16/smp_affinity
$ echo 02〉/proc/irq/18/smp_affinity
$ echo 04〉/proc/irq/20/smp_affinity
```

To set Affinity Mode II, we issue the following two commands:

```
$ echo 01〉/proc/irq/16/smp_affinity
$ echo 02〉/proc/irq/20/smp_affinity
$ echo 04〉/proc/irq/18/smp_affinity
```

Setting the Budget Value. To configure the reception scheme to run in NAPI(300) and NAPI(2), we boot up the system in the default mode, and we set the proper value for the budget to 300 (default) or 2, respectively. The budget is an exposed system variable defined as *netdev_budget*. It can be accessed via the/*proc* virtual file system. The following command can be used to change/view the default budget value to 2, and thereby configuring the reception mechanism to NAPI(2):

```
$ echo 2〉/proc/sys/net/core/netdev_budget
$ cat/proc/sys/net/core/netdev_budget
```

Forwarding Setup. In order to configure the forwarding machine as a packet forwarder as shown in Fig. 4, all packets are sent from the generator machine with a fake same IP address of 10.10.10.10 to both MAC addresses of *eth0* and *eth1*. The forwarding machine has to be configured to route or forward such a packet when received on *eth0* and *eth1*. This can be done by modifying the routing table of the forwarding machine as follows:

```
$ route add -net 10.10.10.0 netmask 255.255.255.0 eth0
$ route add -net 10.10.10.0 netmask 255.255.255.0 eth1
```

Also an ARP entry has to be added to the ARP cache of the forwarding machine. If not, the forwarding machine sends a broadcast ARP request to get the MAC address for IP address 10.10.10.10. Since there is no such host on the network, no response returns, and consequently the forwarding operation fails. An ARP entry can be added as follows:

```
$ arp -s 10.10.10.10 〈eth2 MAC address〉
```

After configuring the routing table of the forwarding machine, the IP forwarding feature has to be enabled. IP forwarding is accomplished by setting the value of/proc/*ip_forward* to 1. The *echo* command can be used to set the value as follows "$ echo 1 〉/proc/sys/net/ipv4/ip_forward".

Disabling Flow Control. In order to be able to generate high traffic rates, it is important to disable Ethernet flow control on the Linux machine for all three network interfaces of *eth0*, *eth1*, and *eth2*. The intent of Ethernet flow control is to prevent packet loss by providing back pressure to the sending NIC to throttle incoming high traffic. However, our experimental work is based on overwhelming the forwarding machine with very high traffic, and thus does not require flow control. Disabling Ethernet flow control can be done by using the "ethtool" Linux utility. Specifically to turn off flow control for *eth0* interface, we issue the command "ethtool -A eth0 rx off tx off autoneg off", and to verify, we issue the command "ethtool -a eth0".

Minimizing Background Activities. To minimize the impact of other system activities on the performance and measurement, we make sure no unnecessary services are running in the background. We boot up the forwarding machine with run level 3, which means the system runs in a single user mode and with fewer services. Run level 3 provides no GUI desktop interface and only shell interface is available. To permanently make the system boot at run level 3, the/etc/inittab configuration file has to be modified.

4. Performance evaluation and comparison

In this section, we report and compare IP forwarding performance in terms of various key metrics Measurements are reported and compared for the two affinity modes under the three reception mechanisms of NAPI(300), NAPI(2), and DE. We also consider measurements using incoming traffic of fixed- and variable-sized packets.

4.1. Performance metrics

We consider a number of key performance metrics for measuring and comparing the performance of IP forwarding. Measuring these metrics would aid not only in evaluating and comparing performance, but also in offering sound interpretations for the behavior of the packet reception mechanisms when used under the two affinity modes. The performance metrics we measure are as follows:

Throughput measures how many packets per second the forwarding machine can sustain. **Packet Loss** is related to the throughput and is basically computed by the difference of the total number of packets sent and received by IXIA divided by the total of packets sent by IXIA. **Round-Trip Delay** is the average delay a packet encounters from the time it is sent by IXIA traffic generator until it is received back. It is to be noted that this delay consists of primarily the forwarding delay at the Linux forwarding machine, as well as other minor delays which include IXIA transmission and reception delays, and cable transmission and propagation delays. These IXIA and cable delays are minor and almost constant for each packet, as packets are generated and timestamped by IXIA at the hardware level, with no software involvement. **CPU availability** is the percentage of idleness of the CPU, which is basically (1—CPU utilization). This metric indicates the amount of CPU cycles left for processing other kernel and user tasks not involved in IP forwarding. **Interrupt Rate or frequency** measures the rate of interrupts for receiving and transmitting packets generated by both Tx and Rx NICs. This aids in understanding the source of overhead and gives insights into the performance behavior of packet reception mechanisms. For measuring CPU availability and interrupt rate, we used the "*sar*" Linux utility at the Linux forwarding machine.

4.2. Performance using fixed-size packets

In this section, we measure the performance by generating the maximum possible packet rate that can be achieved by IXIA based on our experiment setup of Fig. 4. The rate is a constant rate of fixed-size packets with the minimum size of 64 bytes. The maximum packet rate per receiving NIC that can be achieved is around 700 Kpps, which is actually half of the maximum packet rate of 1.4 Mpps that can be sent with 1 Gbps Ethernet link. As discussed in Section 3.1, we used a network tap to send the 1 Gbps generated traffic to both receiving NICs by using two interleaving flows with each flow sending packets to one receiving NIC MAC address.

For all the figures presented in the paper, we denote the symbols of I and II to represent Affinity Mode I and Affinity Mode II, respectively. Now, the throughput and packet loss curves of Affinity Mode I are shown in Fig. 5(a-I) and (b-I), respectively. The throughput and packet loss counterpart curves of Affinity Mode II are shown in Fig. 5(a-II) and (b-II). It is clear that the packet loss curves behave in accordance with throughput curves, satisfying the expected relation $\text{PacketLoss} = (1 - \text{Throughput}/\text{IncomingRate})$. The curves are shown for all three types of packet reception mechanisms. The figure shows that the average throughput of IP forwarding using Affinity Mode II has more throughput than that of Affinity Mode I of around 20 Kpps for the three types of packet reception mechanisms. This is clearly noticeable under heavy traffic load conditions, i.e., when traffic rate goes beyond 250 Kpps.

The throughput gain in the case of Affinity Mode II is primarily attributed to the fact that heavy traffic load has resulted in less LLTX lock contention than in Affinity Mode I where Tx API task encounters more overhead in acquiring the LLTX lock. The overhead of the Tx API is evident in Fig. 8(b-I) and (b-II) which shows the CPU availability of the core affinitized with the Tx API core is smaller in Affinity Mode I than in Affinity Mode I. In addition, it is obvious that in Affinity Mode II, the L2 cache utilization is better than in Affinity Mode I, as both cores affinitized with the Tx and Rx NIC 1 share the same L2 cache. In this situation, data access to packet descriptors to be transferred from one RxRing to the TxRing is faster than Affinity Mode I, as the L2 cache of CPU 0 is being utilized by the concurrent SoftIRQs being run by the two cores located on the same processor. On the other hand, slow memory access is encountered in the case of Affinity Mode I, as the L2 cache is not utilized (and in fact being invalidated) since the two RxRings and TxRing are affinitized to separate cores located on separate processors, each with its own L2 cache. Fig. 5 shows a difference in throughput, in the heavy load region among the three types of reception mechanisms, with the highest performance being achieved with NAPI(300) and DE in the case of Affinity Mode II, and DE in the case of Affinity Mode I. This performance is attributed to the reception mechanism algorithm in acquiring and releasing the LLTX lock, as will be explained next in the interpretation of the delay curves of Fig. 6.

Fig. 6 shows the average packet round-trip delay. Fig. 6(b-I) and (b-II) are a zoom-in version of Fig. 6(a-I) and (a-II), respectively, showing and comparing delays at low incoming traffic rates. As shown, Affinity Mode II yields less delay than Affinity Mode I, specifically under heavy traffic load conditions. Surprisingly at low traffic load conditions, both affinity modes exhibit similar round-trip delays. The interpretation of the delay curves are attributed to the four key software and hardware architectural bottlenecks (discussed in detail in Section 2.3): (1) difference in computational cost and behavior among the packet reception mechanisms, (2) "LLTX" lock contention for guaranteeing the TxRing structure is accessed by the parallel SoftIRQs in a mutually exclusive fashion, (3) L2 cache utilization and invalidation, and (4) the memory and I/O bandwidth increase in the case of Affinity Mode II when the cores do not share the FSB bus.

At low traffic rate (less than 200 Kpps), the dominating factor for the delay performance is the computational cost of the packet reception mechanism. Fig. 6(b-I) and (b-II) show that the DE scheme outperforms other reception mechanisms. As stated in Section 2.1 and as reported in Salah and Qahtan (2009), DE does not defer packet handling and processing, as in the case of NAPI. DE processes packets exhaustively at the interrupt level, whereas NAPI at the interrupt level only notifies the high-priority kernel function of SoftIRQ, thereby deferring packet processing until SoftIRQ is scheduled to run at a later point by the kernel scheduler. As shown in Figs. 6(b-I) and (b-II), the delay of NAPI(2) is smaller than that of NAPI(300), as NAPI(2) processes limited number of packets in a single SoftIRQ invocation, thereby releasing "LLTX" lock much sooner than NAPI(300). It is observed from Fig. 6(b-I) and (b-II) that a short increase in the delay occurs specifically around an incoming traffic of 100 Kpps. This short increase is then followed by a short fall. Due to scaling, such behavior is not seen in Fig. 6(a-I) and (a-II). Such behavior is due primarily to the incurred interrupt processing overhead around an incoming rate of 100 Kpps, as shown clearly in Fig. 8(a-I) and (a-II). An interpretation of the interrupt curves will be given when discussing Fig. 8.

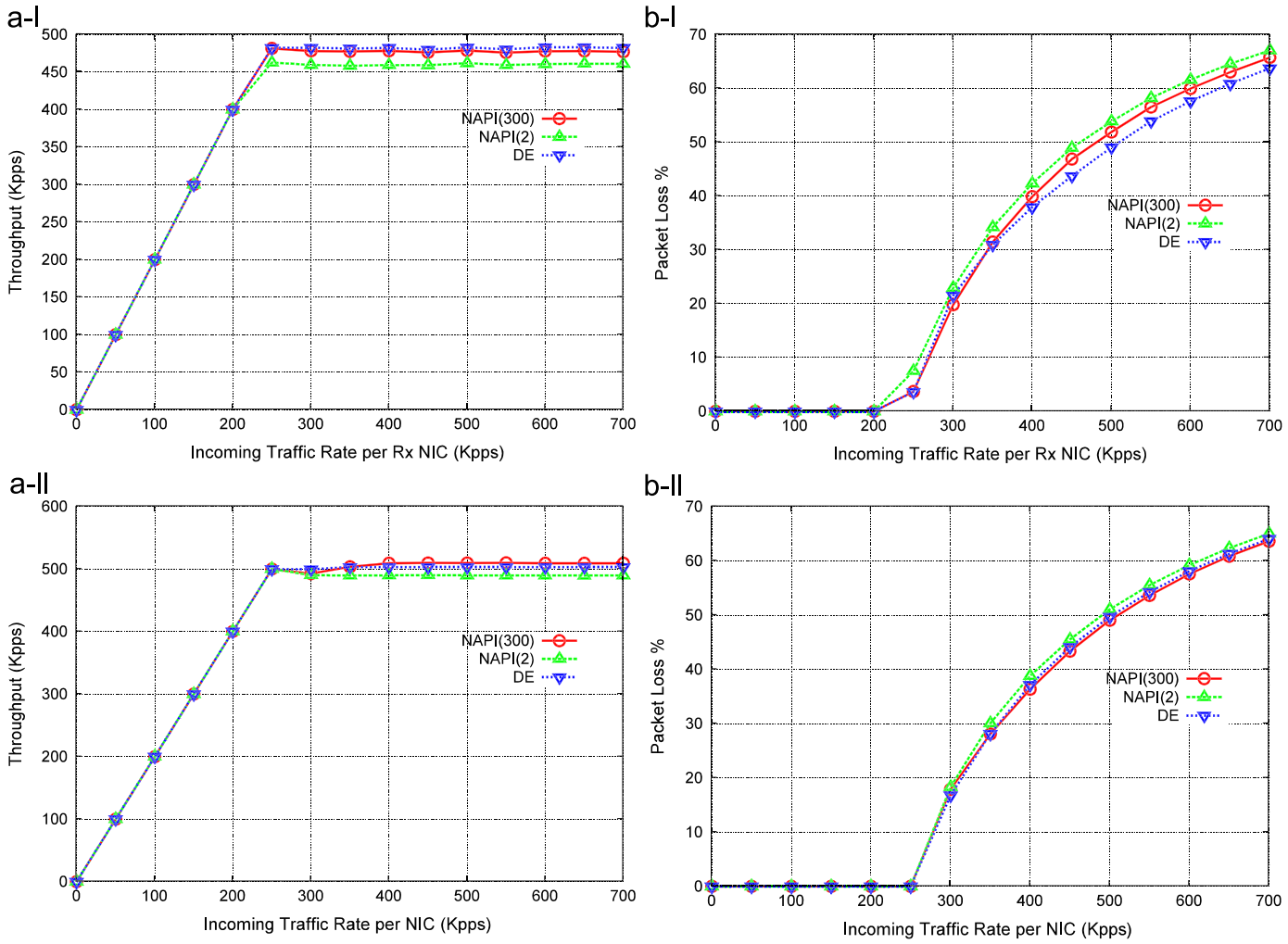

Fig. 5. Throughput and corresponding packet loss.

At heavy traffic load conditions, Fig. 6(a-I) and (a-II) show that Affinity Mode II gives less delay than Affinity Mode II. The primary reason for this increase in the delay is because heavy traffic load has resulted in substantial demand for accessing L2 cache and requiring LLTX lock. In Affinity Mode I, memory accesses to manipulate and transfer packets from the RxRing to the shared TxRing become slow (when compared with Affinity Mode II), as the L2 cache is being utilized poorly in transferring packets from both RxRings to the shared TxRing. In fact, the L2 cache is being invalidated and is encountering severe cache bouncing, as discussed in Section 2.3. It is observed from Fig. (a-I) and (a-II) that the delay curves for DE exhibit the smallest delay. The reason is because DE introduces less overhead in processing packets than NAPI, and also DE processing of packets is done at interrupt level. It is to be noted that the separation of FSB in Affinity Mode II had little influence on the performance results. This is expected as the FSB runs at a speed of 1333 MHz and is capable of transferring up to 21 GB/s of aggregated throughput (as stated in Section 2.2). Such aggregated throughput is more than adequate for accessing and manipulating packets arriving at a rate of 1 Gbps.

The CPU availability of cores affinitized with Rx NIC 1 and Rx NIC 2 are shown in Fig. 7. The curves of CPU availability of cores which are affinitized with the Rx NIC 1 and Tx NIC are shown in Fig. 8. Fig. 7 shows that CPU availability curves for both receiving rings under the three reception mechanisms are identical. This is expected since both receive identical traffic, and are symmetric in design. Fig. 8 shows that the Rx CPU availability is much smaller than that of the Tx CPU availability. The figure confirms (as stated in the discussion of Section 2) that most of the computational cost involved in IP forwarding is associated with the reception mechanism which is handling and processing receiving packets, i.e., Rx API and packet forwarding. The computational cost of Tx API is far less than the cost of the transmission mechanism, as the task of Tx API has limited responsibility for simply moving packets from TxRing into the NIC buffer for transmission and then de-allocating all already transmitted packets from TxRing. For ease of comparison and presentation, it is to be noted that Fig. 7(a-I) is a duplicate of Fig. 8(a-I).

The curves for CPU availability of cores affinitized with Rx NICs in both affinity modes are similar in behavior, as shown in Fig. 8(a-I) and (a-II). However, there is more Rx CPU availability in the case of Affinity Mode II for all reception mechanisms, particularly when incoming traffic rate is less than 350 Kpps. This availability increase is due to more utilization of the L2 Cache between the two cores of CPU 0, resulting in speeding up the data access and reducing the processing time being spent in spin locking to acquire "LLTX" lock. When comparing Rx CPU availability among reception mechanisms, it is observed that the availability is slightly greater in the case of DE than NAPI(300) or NAPI(2). The reason is the higher computational cost of NAPI compared with that of DE (as discussed in Section 2.3).

When comparing the Tx CPU availability curves of Affinity Mode I to its respective curves of Affinity mode II, as showin in

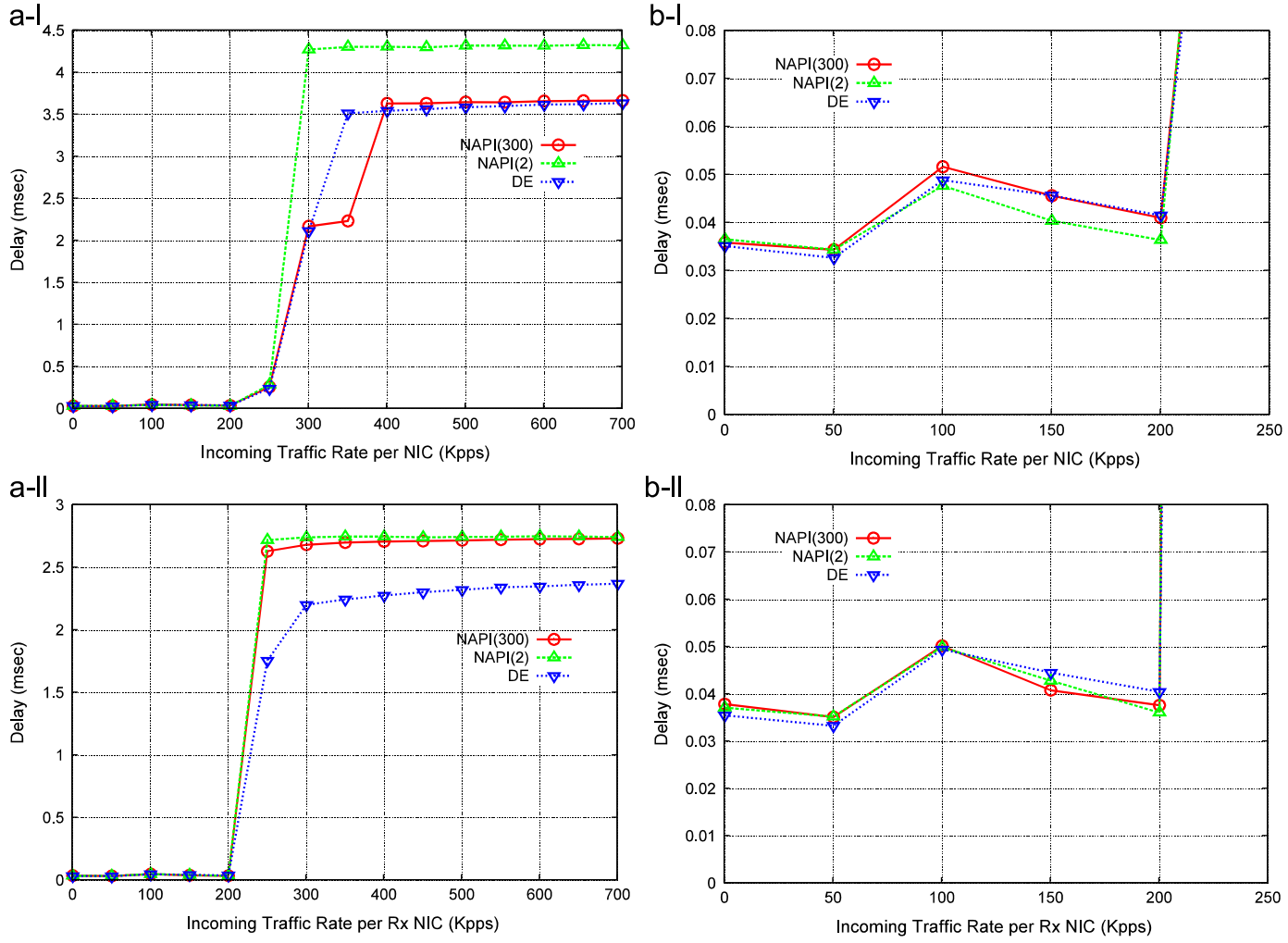

Fig. 6. Round-trip delay per packet.

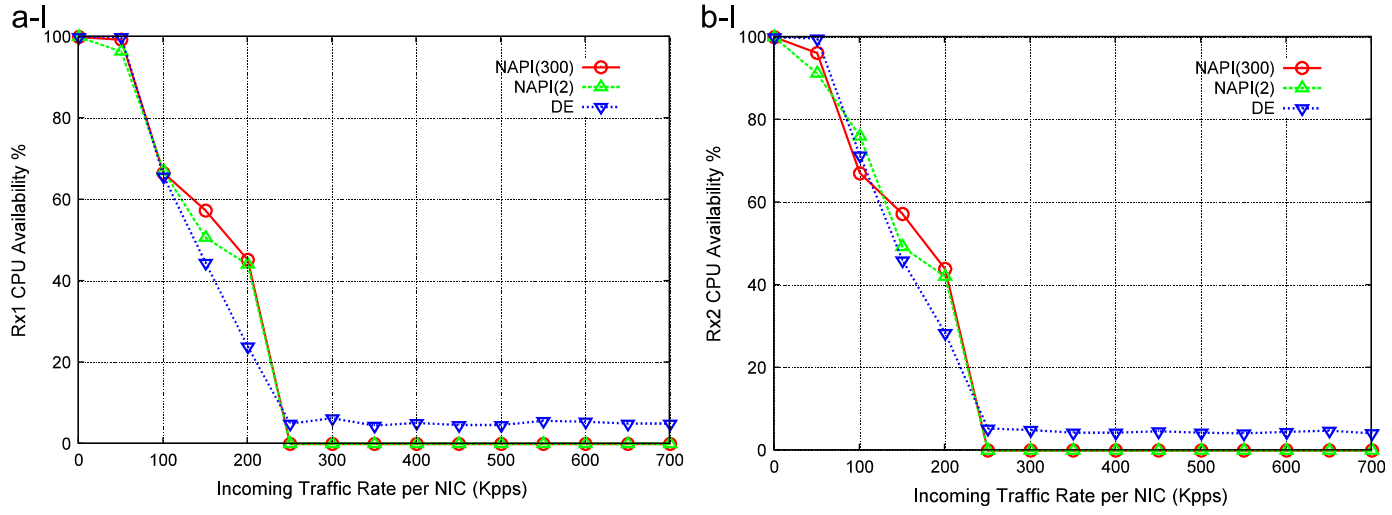

Fig. 7. Comparison of CPU availability of Rx1 and Rx2 CPUs in Affinity Mode I.

Fig. 8(b-I) and (b-II), it is clear that Affinity Mode II yields much higher CPU availability. For example, at high rate, Tx CPU availability using Affinity Mode I is around 60%, whereas it is around 40% using Affinity Mode II. This can be largely attributed to the fast data access to packet descriptors between RxRing of NIC 1 and TxRing due to high utilization of L2 cache, thereby speeding up the movement of packets between these two rings in addition to boosting the speed of acquiring and releasing LLTX lock. In Affinity Mode I, slow data access is presented via FSB and RAM to acquire and release the LLTX lock with no utilization of L2 Caches.

Fig. 9 plots the interrupt rates of Rx1 and Rx2 NICs in Affinity Mode I. Fig. 10 plots Rx1 and Tx interrupt rates in relation to

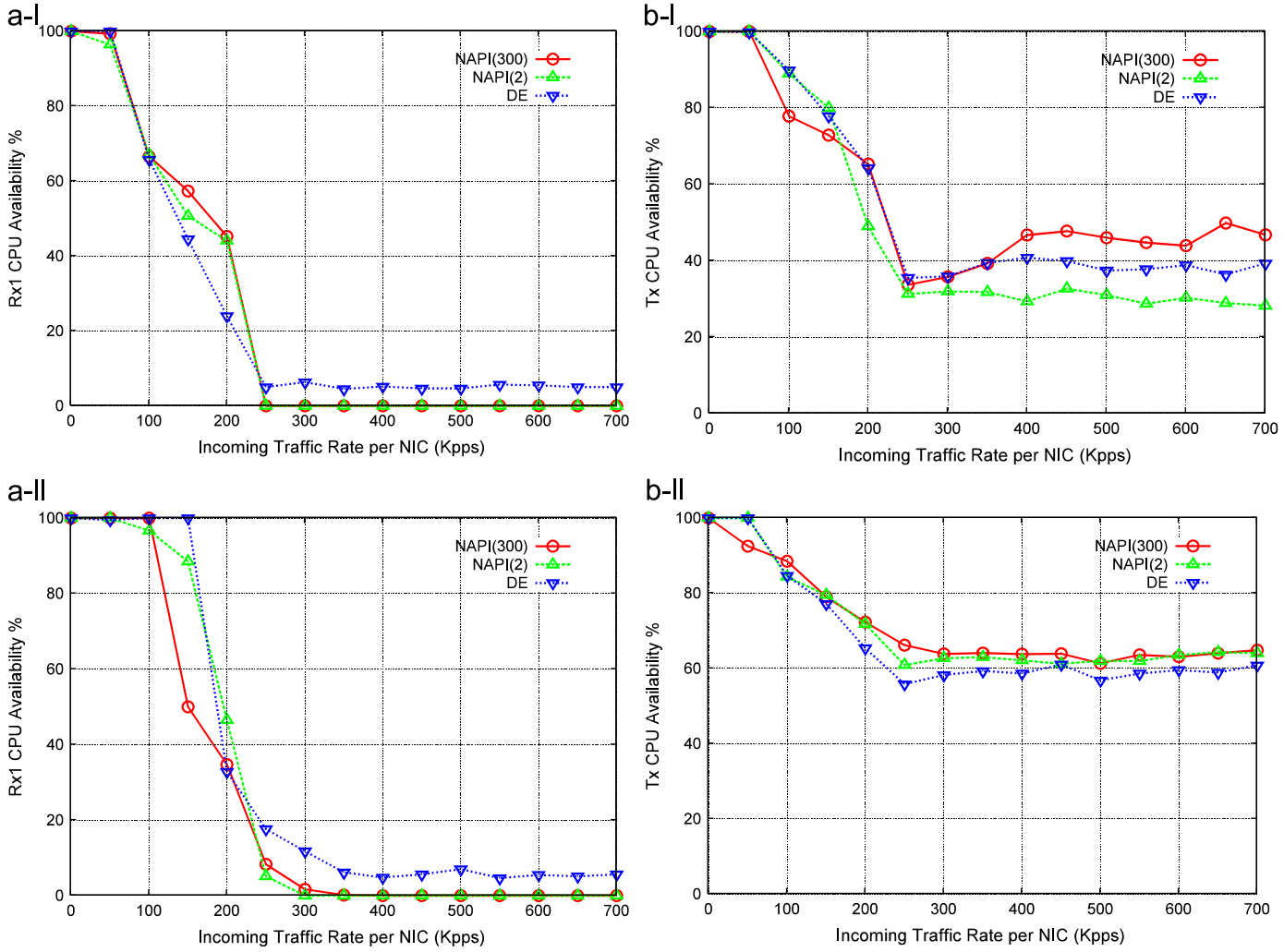

Fig. 8. CPU availability of cores affinitized with Rx1 and Tx NICs for Affinity Mode I and II.

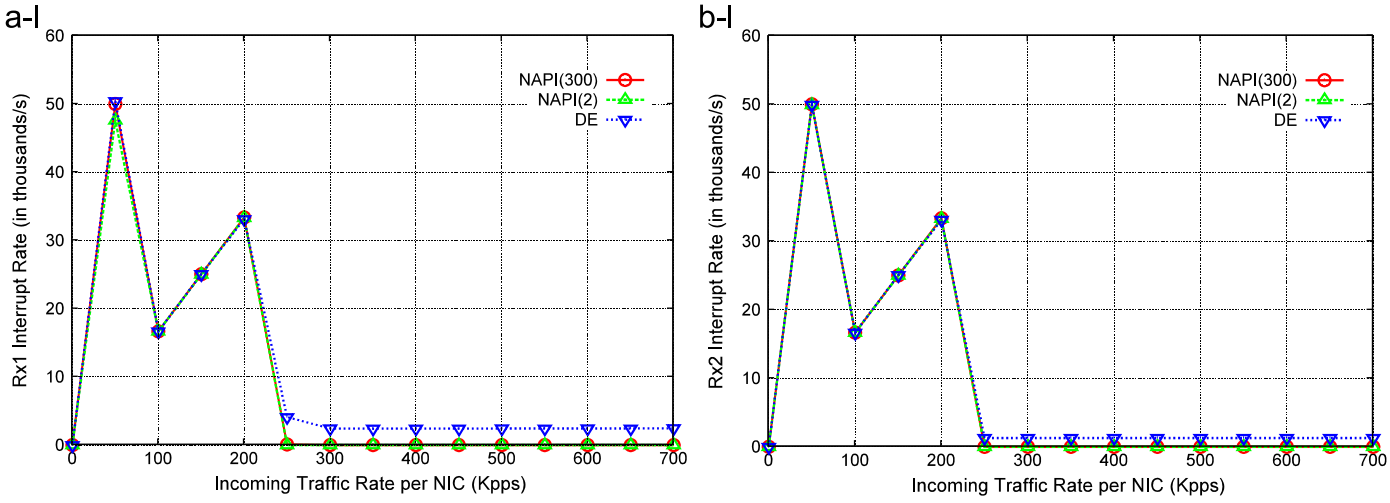

Fig. 9. Comparison of Interrupt rates of Rx1 and Rx2 NICs in Affinity Mode I.

incoming traffic rate. The plots give more insights into the behavior and dynamics of reception mechanisms, and better interpretations of the performance curves of throughput, latency and CPU availability. Fig. 9 confirms that the interrupt rates introduced by both receive NICs are similar. From Fig. 10, it is observed that Rx1 interrupt rate of Affinity Mode I resembles that of Affinity Mode II, with Affinity Mode I showing smaller Rx interrupt rate, particularly under high traffic loads of the region between 200 and 350 Kpps. The primary reason for reduction in interrupt rates in Affinity Mode I is that the computational or processing time of received packets in Affinity Mode I is larger than in Affinity Mode II, which was evident from Fig. 8(a-I) and

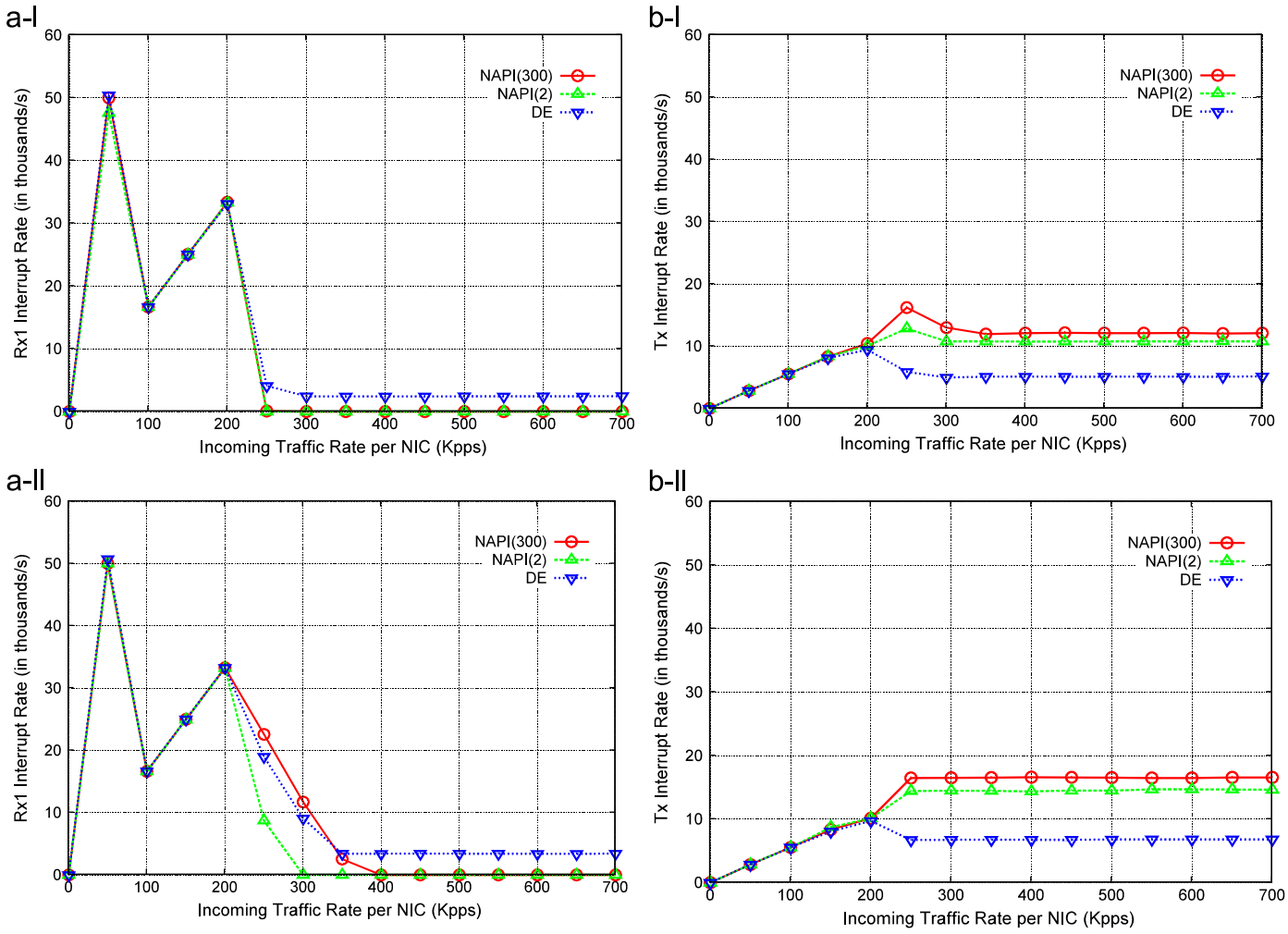

Fig. 10. Interrupt rates of Rx1 and Tx NICs.

(a-II), whereby the CPU availability for Mode II is slightly larger than Mode I, due to L2 cache invalidation and bouncing. It should be noted that during the processing time of received packets (for all reception mechanisms), Rx interrupts are disabled (or masked). Since the processing time period in Mode I is more than in Mode II, Rx interrupt rates in Mode I will be slower than those of Mode I, as the chance of interrupts being masked during longer processing times will increase.

Fig. 10(a-I) and (a-II) exhibit the impact of the three reception schemes on the mitigation the Rx interrupt rate. It is observed that, under a rate below 50 Kpps, the Rx interrupt rate increases linearly in relation with the arrival rate for all three schemes. Shortly after that, the Rx interrupt rate drops significantly, and then it starts increasing again. This is in line with the expected behavior. Below 50 Kpps, the Rx interrupt rate is low and the host is able to finish the interrupt handling and processing of packets before the next interrupt. The period of disabling and enabling Rx interrupts (i.e., interrupt masking period) for all of the three schemes 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 the Rx interrupt rate shortly after 50 Kpps. After 60 Kpps the interrupt rate starts increasing again with respect to the arrival rate. Rx interrupt masking still occurs, but the masking period is not stretched considerably with the higher rate. The CPU still has the power to handle and process packets relatively quickly.

At an incoming rate of around 200 Kpps, it is observed that the Rx interrupt rates for NAPI(300) and NAPI(2) fall to almost zero, with NAPI(2) curve falling earlier than NAPI(300). The reason for this fall is mainly the inability to re-enable interrupts as NAPI is not able to finish (or exhaust) processing all packets in one polling cycle (or in one SoftIRQ invocation) at such high rates. The Rx interrupt rate for NAPI(2) falls earlier than NAPI(300) because the masking period for NAPI(2) stretches much sooner under such a high rate. Unlike NAPI(300), NAPI(2) would not be able to exhaust all packets soon enough in one polling period to be able to re-enable Rx interrupts. On the other hand, the masking period for DE is much smaller than NAPI(300) and NAPI(2) as it requires less computational cost and is giving higher priority processing. Even at a high incoming traffic rate, DE is always able to finish processing packets and then re-enabling interrupts, resulting in a smaller but not completely eliminated Rx interrupt rate.

Fig. 10(b-I) and (b-II) show that the Tx interrupt rates for Affinity Mode II for all three reception schemes are slightly higher than those rates for Affinity Mode I. This is expected and in line with the throughput curves exhibited for both affinity modes in Fig. 5(a-I) and (a-II), whereby throughput curves for Affinity Mode II are higher than Affinity Mode I. Tx interrupt rates are highly affected by two primary factors: (1) the speed of the reception mechanism to process received packets and place them in the shared TxRing, and (2) the speed of the Tx API transmission mechanism to access packets from the shared TxRing and move them to the Tx NIC. In other words, the dominating factor is

attributed to the utilization of the L2 cache which speeds up access to TxRing by both reception and transmission mechanisms and also reduces LLTX lock contention. The LLTX lock contention plays a major role for both NAPI(300) and DE. In Affinity Mode II, the curves of Tx interrupt rate for NAPI(300) and NAPI (2) are higher than in Affinity Mode I by almost 4000. Similarly, a difference of almost 2000 is exhibited for the DE reception mechanism.

4.3. Performance using variable-size packets

In this section, we measure and compare the IP-forwarding performance by subjecting the forwarding host to traffic of variable-size packets instead of fixed-size packets. Both traffic types of fixed- and variable-size packets follow standard evaluation methodology of RFC2544 to assess network performance of servers, gateways, routers, and switches (Brander and McQuaid, 1999). While a traffic of fixed-size packets of 64 bytes is important in examining performance under the most severe traffic load condition, a traffic with variable-size packets is closer to real-world network traffic, according to empirical measurements of packet sizes taken from MCI backbone (Sinha, Papadopoulos and Heidemann) and available online at http://www.caida.org. Statistical analysis shows that dominating packet sizes are 64, 594, and 1518 bytes. Therefore, we configure IXIA to generate traffic flows with packets that follow these dominating sizes. From IxExplorer, we select a predefined distribution (called the Cisco Distribution) which is preconfigured to generate the flows with variable-size packets. Specifically, Cisco Distribution generates traffic flows with packets of the three sizes 64, 594, and 1518 bytes. These packets are generated with weights of 7, 4, and 1, respectively. As in the case of fixed-sized packets, we used the network tap to relay the 1 Gbps generated traffic to both receiving NICs by using two interleaving flows with each flow directing packets to one receiving NIC MAC address.

Fig. 11 plots the throughput and corresponding packet loss when the packets of incoming traffic are variable in size. It is noticed that with the predefined packet-size Cisco Distribution, the maximum incoming rate that can be generated by IXIA is 320 Kpps, and the effective packet rate being received per NIC is 160 Kpps. This rate is considerably smaller than the rate of 1400 Kpps (or 700 Kpps for the split generated traffic using the CC1200-P network tap) which was achieved when using fixed-size packets with the minimum size of 64 bytes. The slowdown of the rate is expected as variable-size packets would stretch the inter-arrival times of packets, thereby reducing the effective generated traffic rate. Fig. 11 shows that the throughput and the corresponding packet loss for all three reception schemes in are similar, with Affinity Mode II at heavy traffic load exhibiting slightly higher throughput than Affinity Mode I, for NAPI(2) and NAPI(300).

An important observation can be made when comparing the saturation point of throughput of fixed- and variable-sized packets. For example, Fig. 11(a) shows that the saturation point (or highest achievable capacity or throughput) dropped to an aggregated transmitting throughput of 280 Kpps from its respective saturation point

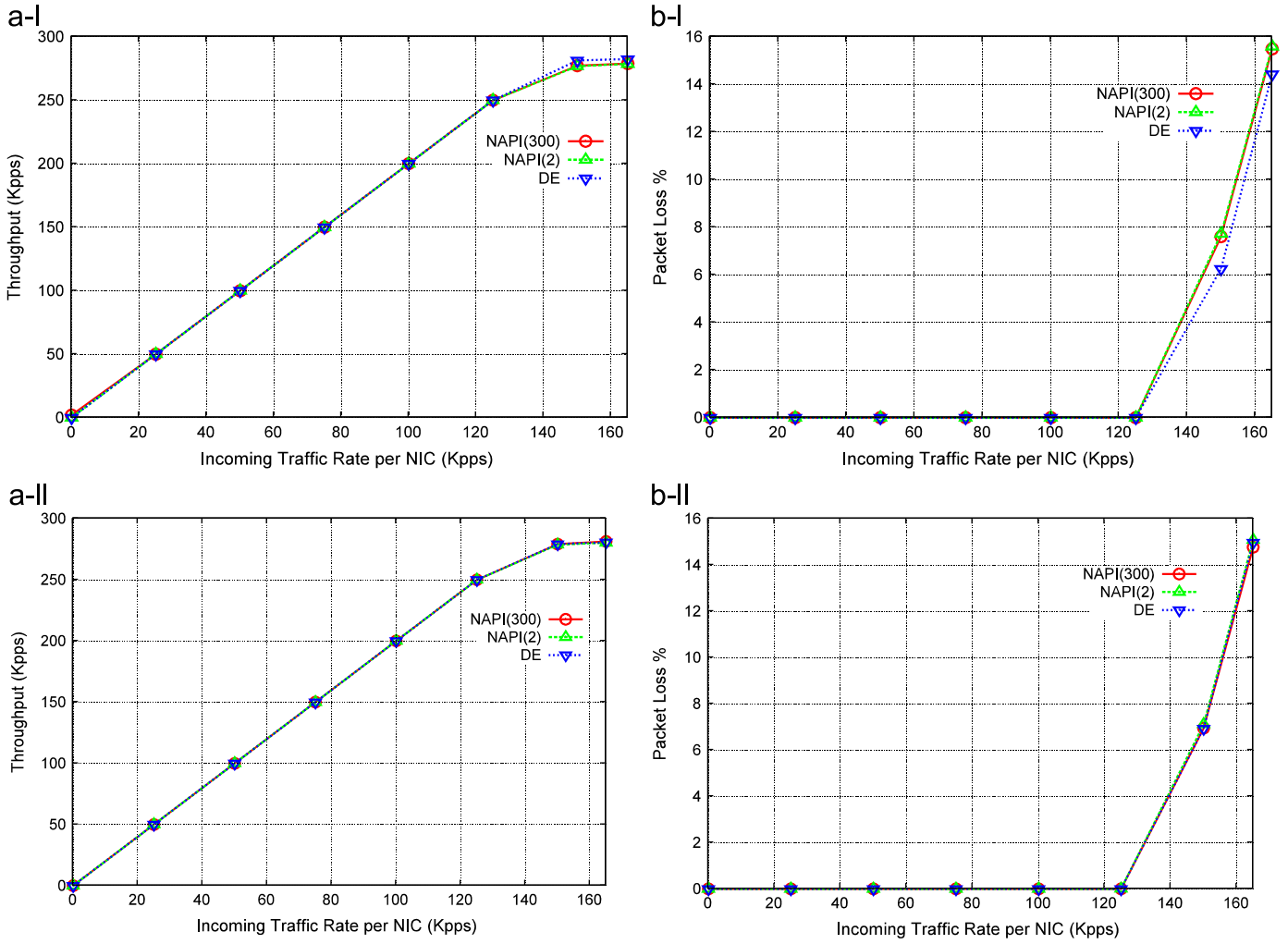

Fig. 11. Throughput and corresponding packet loss when using variable-size packets.

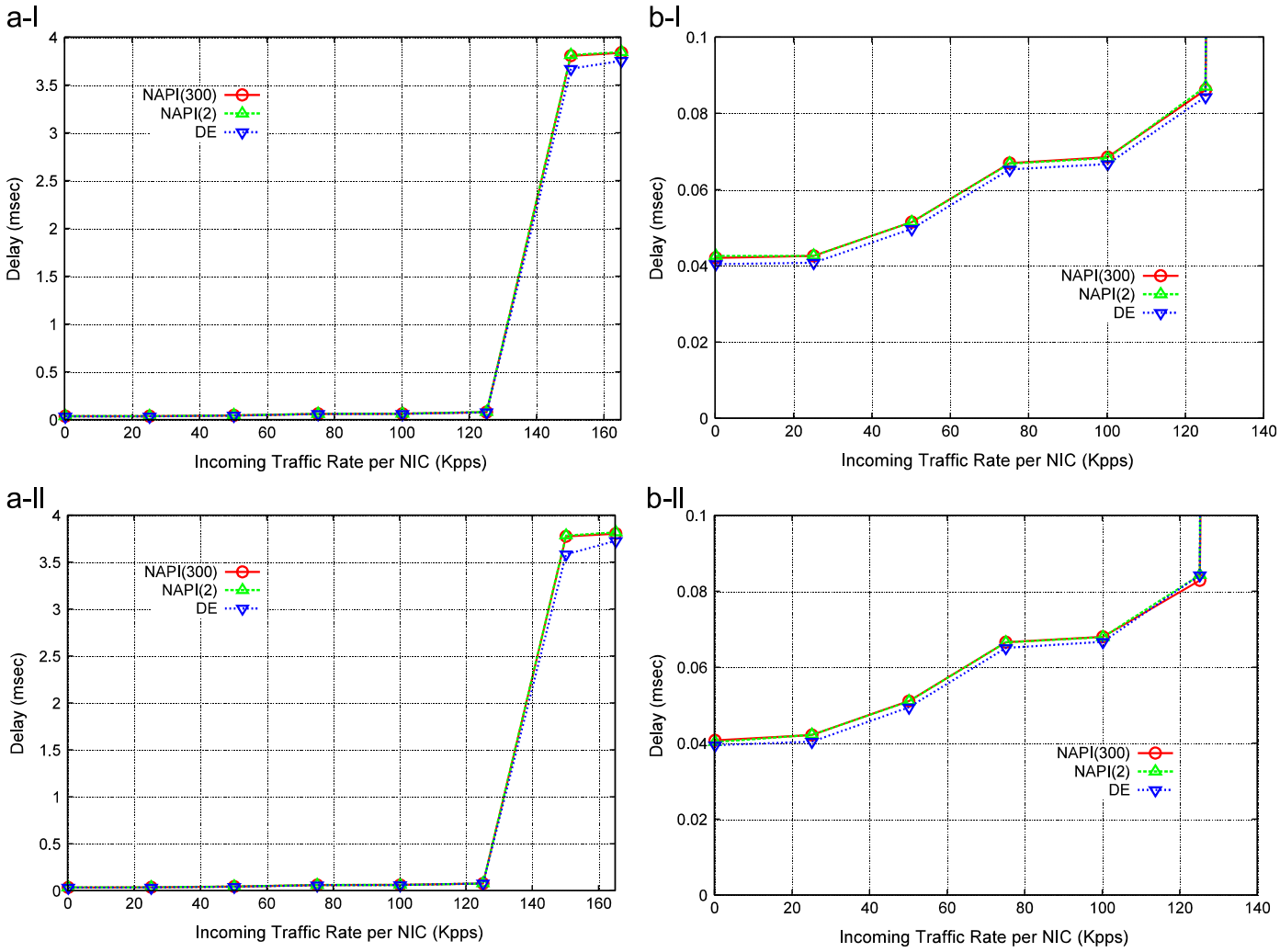

Fig. 12. Round-trip delay per packet when using variable-size packets.

of around an aggregated transmitting throughput of 500 Kpps shown in Fig. 5(a). For a single incoming rate, transmitting throughputs of 140 and 250 Kpps can be achieved for variable-size packet traffic and fixed-sized packet traffic, respectively. This significant drop in throughput capacity is attributed to the fact that the variable-size packets increased a number of system delays and times which include: (1) queueing delays of on-board NIC buffers as well as the kernel's RxRing and TxRing buffers, (2) data transfer time via DMA from NIC to RxRing and from TxRing to DMA, (3) transmission time from the NIC to wire, and (4) NIC offload processing time to verify the checksum of the entire Ethernet frame. These delays also contribute to an increase in the overall round-trip delay, as depicted next in Fig. 10.

The corresponding round-trip delay when using variable-size packets is plotted in Fig. 12. The figure shows the curves of all three reception mechanisms are very similar in both affinity modes. It should be noted that the DE reception mechanism outperforms other mechanisms under low and high traffic rates, and for both affinity modes. However, when comparing the delay exhibited by traffic of fixed-size with variable-size packets (i.e., Fig. 6 with Fig. 12), it is noticed that the delay of variable-size packets increased for most reception mechanisms, and more noticeably when traffic rate is high. For example, Fig. 6(a-II) shows the round-trip delay exhibited at high rate using Affinity Mode II is in the range of 2.3 to 2.7 ms for traffic of fixed-size packets, whereas Fig. 10(a-II) shows a delay of around 3.7 ms for variable-size packets. It is also observed that under low incoming traffic rate, the latency fluctuation exhibited in Fig. 12(b) is not as severe for variable-size packets as for fixed-size. This behavior is expected, as the Rx interrupt rates are not as high with variable-size packets as with fixed-size, thereby resulting in less overhead and latency. This can be confirmed by comparing Fig. 13(a) with Fig. 10(a).

Fig. 13 plots the Rx and Tx interrupt rates with respect to the traffic rate of variable-size packets. One important observation can be made when comparing Fig. 13 to Fig. 10. Fig. 13 shows that the interrupt rates of incoming packets and transmitted packets have similar behavior as those presented in Fig. 10, but have dropped significantly. This is due to using variable-size packets which results in stretching the interarrival times, thereby slowing down the interrupt rates. The curves for Tx interrupt rates of Fig. 13(b-I) and (b-II) exhibit similar behavior to those of Fig. 10(b-I) and (b-II), but with slowdown in interrupt rates. The reception scheme of DE still shows the lowest Tx interrupt rate for traffic of fixed- and variable-size packets.

5. Concluding remarks

We evaluated and compared experimentally the performance of IP forwarding of a typical MCMP Linux host with two receiving NICs and one transmitting NIC. We considered two possible NIC affinity modes. Affinity Mode I affinitizes the two receiving NICs

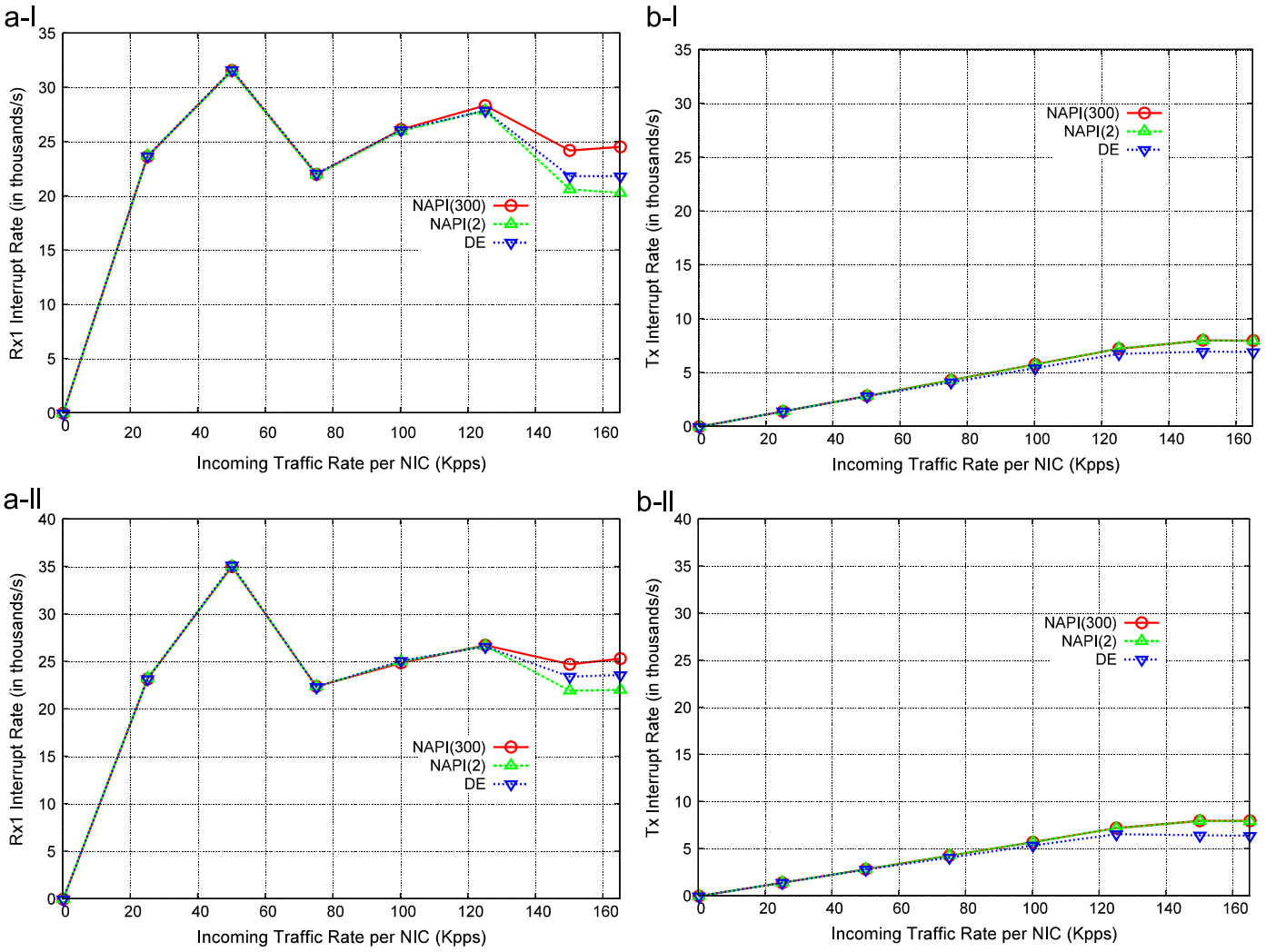

Fig. 13. Interrupt rates of Rx1 and Tx NICs when using variable-size packets.

to two cores of the same processor while the transmitting NIC to a core on a separate processor. Affinity Mode II affinitizes the transmitting NIC and one receiving NIC to two cores of the same processor while the second receiving NIC to a core on a separate processor. The performance measurements for these two affinity modes were done for three packet reception mechanisms, namely NAPI(300), NAPI(2), and DE. We used the IXIA packet generator to generate and analyze packets. We generated traffic with fixed- and variable-size packets. The performance was measured and compared in terms of a number of key metrics which included throughput, packet loss, delay, interrupt rates, and CPU availability. In this paper, we identified four primary factors that can potentially impact the IP-forwarding performance. Our measurements show that the utilization of the L2 cache and the contention for "LLTX" lock were the most dominating factors. The utilization of the L2 cache, in Affinity Mode II, resulted in speeding up the data access to TxRing and reduced the processing time spent in spin locking to acquire "LLTX" lock. We demonstrated that the best performance in terms of throughput, latency, and CPU availability was achieved with our previously proposed and implemented DE reception mechanism (Salah and Qahtan, 2009) when used with Affinity Mode II. Our results show that there was little difference in the shape of performance curves when the forwarding machine is subjected to traffic containing variable-size packets instead of fixed-size packets. However, the traffic of variable-size packets yielded a substantial slowdown in throughput and a substantial increase in latency.

Our measurements for CPU availability showed that the CPU cores affinitized with the receiving NICs were highly utilized when compared to the utilization of the core affinitized with the transmitting NIC, particularly under heavy traffic load conditions. This suggests the reception mechanisms have to be redesigned and parallelized to balance the task load of receiving, forwarding and transmitting, and most importantly should be augmented to take advantage of other available cores of an MCPM host. This improvement is planned as a future study. Also in a future study, we plan to investigate and optimize the performance of hosts with new emerging Intel multicore architecture whereby each core has its own independent FSB and separate L2 cache.

References

Bolla R, Bruschi R IP forwarding performance analysis in presence of control plane functionalities in a PC-based Open Router. Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, Springer, Norwell, MA, 2006, p. 143–158.

Bolla R, Bruschi R. An effective forwarding architecture for SMP Linux routers. Proceedings of the 4th Int. Telecom. Networking Workshop on QoS in Multi-service IP Networks (QoS-IP 2008), Venice, Italy, 2008a, p. 210–216.

Bolla R, Bruschi R. PC-based Software Routers: high performance and application service support. in: Proceedings of the ACM Workshop on Programmable Routers For Extensible Services of Tomorrow, August 2008b.

Bovet D, Cesati M. Understanding the Linux Kernel. 3rd Edition O'Riley Press; 2005.
Brander S, McQuaid J. RFC2544 – Benchmarking Methodology for Network Interconnect Devices, March 1999.
Chen B, Morris R. Flexible control of parallelism in a multiprocessor PC router. in: Proceedings of the General Track: 2002 USENIX Annual Technical Conference, June 2001.
Emma, Pescape A, Ventre G, "D-ITG, Distributed Internet Traffic Generator", 2004, Available from 〈http://www.grid.unina.it/software/ITG〉.
Foong A, Fung J, Newell D, Abraham S, Irelan P, Lopez-Estrada A. Architectural characterization of processor affinity in network processing, Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, March 2005, p. 207–218.
Guagga, "Guagga Routing Suite", Available from 〈http://www.quagga.net〉.
Handley M, Hodson O, Kohler E. ORP: an open platform for network research. ACM SIGCOMM Computer Communication Review 2003;33(1):53–7.
Huang T, Zeadally S, Chilamkurti N, Shieh C. Design, implementation, and evaluation of a programmable bandwidth aggregation system for home networks. Journal of Network and Computer Applications 2009;32(3):741–59.
IXIA Inc., "Product Description of IXIA 400T Traffic Generator/Performance Analyzer with LM1000SX Module", Available from 〈http://www.ixiacom.com/products/chassis/disp〉.
IXIA Inc., "Product Description of IxExplorer GUI for Traffic Generation and Analysis", Available from 〈http://www.ixiacom.com/products/display?skey=ixexplorer〉.
Leroux P. Meeting the bandwidth challenge: building scalable networking equipment using SMP. Dedicated Systems Magazine 2001.
Markatos E, LeBlanc T. Using processor affinity in loops scheduling on shared-memory multiprocessors. in: Proceedings. of the 1992 ACM/IEEE Conference on Supercomputing, 1992, p. 104–113.
Microsoft Inc., "Scalable networking: eliminating the receive processing bottleneck—introducing RSS", Microsoft whitepaper, available at 〈http://www.microsoft.com/whdc/〉, 2004.
Mogul J, Ramakrishnan K. Eliminating receive livelock in an Iiterrupt-riven kernel. ACM Trans Computer Systems 1997;15(3):217–52.
Mohamed N, Al-Jaroodi J. Self-configured multiple-network-interface socket. Journal of Network and Computer Applications 2010;33(1):35–42.
Morris R, Kohler E, Jannotti J, Kaashoek M. The click modular router. ACM Transactions on Computer Systems 2000;8(3):263–97.
NetworkCritical Inc., "100/1000 Critical Tap User Guide", Available from 〈http://www.networkcritical.com/NetworkCritical/files/03/0355d462-694d-4962-906c-0a92079e95a9.pdf〉.
Nortel Inc, "Contivity VPN switches", Available from 〈http://www.nortel.com/products/01/contivity/techspec.html〉.
Olsson R. "Pktgen the Linux Packet Generator", In the proceedings of Linux Symposium, Ottawa, Canada, 2005.
Olsson R, Wassen H, Pedersen E. Open Source Routing in High-Speed Production use, Unpublished, 2008.
Salah K, Hamawi M. Comparative packet-forwarding measurement of three popular operating systems. International Journal of Network and Computer Applications, Elsevier Science 2009;32(5):1039–48.
Salah K, Kahtani A. Performance evaluation comparison of snort nids under linux and windows server. International Journal of Network and Computer Applications, Elsevier Science 2010;33(1):6–15.
Salah K, Qahtan A. Implementation and experimental performance evaluation of a hybrid Interrupt-Handling scheme. International Journal of Computer Communications, Elsevier Science 2009;32(1):179–88.
Salah K, El-Badawi K, Haidari F. Performance analysis and comparison of Interrupt-Handling schemes in Gigabit Networks. International Journal of Computer Communications, Elsevier Science, Computer Communication 2007;30(17):3425–41.
Salah K, Hamawi M, Hassan Y. On the performance of IP-forwarding for multicore multiprocessor linux hosts. International Journal of IET Communications 2010;4(18):2166–80.
Salim JH, Olsson R, Kuznetsov A. "Beyond Softnet," Proceedings of the 5th Annual Linux Showcase and Conference, November 2001, pp. 165–172.
Sinha R, Papadopoulos C, Heidemann J. Internet Packet Size Distributions: Some Observations, University of Southern California, Technical Report ISI-TR-2007-643, May 2007, Available at 〈http://www.isi.edu/~johnh/PAPERS/Sinha07a.html〉.
Siris VA, Stavrakis I. Provider-based deterministic packet marking against distributed DoS attacks. Journal of Network and Computer Applications 2007;30:858–76.
Subramaniam S, Eager D. Affinity scheduling of unbalanced workloads. Supercomputing 1994;94:214–26.
White R, Bollapragada V, Murphy C. Inside Cisco IOS Software Architecture. Cisco Press; 2000.
Xorp Org., "Open Source IP Router", Available from 〈http://www.xorp.org/〉.
Zander S, Kennedy D, Armitage G. "KUTE—A High Performance Kernel-based UDP Traffic Engine"; CAIA (Center for Advanced Internet Architectures) Technical Report (2005), Available from 〈http://caia.swin.edu.au/reports/050118A/CAIA-TR-050118A.pdf〉.
Zebra Inc, "GNU Zebra – Routing Software", Available from 〈http://www.zebra.org/〉.