Click vs. Linux: Two Efficient Open-Source IP Network Stacks for Software Routers

Andrea Bianco*, Robert Birke*, Davide Bolognesi*, Jorge M. Finochietto*, Giulio Galante†, Marco Mellia*, Prashant M.L.N.P.P.‡, Fabio Neri*

* Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy, Email: {bianco, finochietto, mellia, neri}@polito.it

† Networking Lab, Istituto Superiore Mario Boella, 10138 Torino, Italy, Email: galante@ismb.it

‡ Indian Institute Of Technology, Guwahati, India, Email: prasanth@iitg.ernet.in

***Abstract*— Software routers based on off-the-shelf hardware and open-source operating systems are gaining more and more momentum. The reasons are manifold: first, personal computer (PC) hardware is broadly available at low cost; second, large-scale production and the huge market spur the manufacturers to closely track the improvements made available by Moore's Law; third, open-source software leaves the freedom to study the source code, learn from it, modify it to improve the performance, and tailor its operation to one's own needs. In this paper we focus only on the data plane performance and compare the default Linux IP stack with the Click modular IP stack in terms of the forwarding throughput. The results are surprising and show that a high-end PC is easily able to fit into the multi-gigabit-per-second routing segment, for a price much lower than commercial routers.**

I. Introduction

Routers are the glue keeping IP packet networks together, and have always received a lot of attention from both the academic and the industrial community.

Traditionally, high-end networking equipment has always been built with custom application specific integrated circuit (ASIC) components for efficiency reasons. Unfortunately, equipment based on proprietary hardware and closed-source software usually has several drawbacks. First, hardware produced by different vendors may interoperate only partly or even not interoperate at all. Second, the unavailability of the source code of the software and the lack of documentation on the hardware, often make fixing problems and introducing new features difficult. Third, last but not least, the price at which commercial networking equipment is sold is often much higher than its actual value.

On the contrary, software routers based on off-the-shelf PC hardware and open-source software are becoming appealing alternatives to proprietary network devices because of *i*) the wide availability of multi-vendor hardware and documentation on their architecture and operations, *ii*) the low cost, and *iii*) the continuous evolution driven by the PC market's economy of scale. Indeed, the PC world benefits from both the de-facto standards defined for hardware components, which enable the development of an open multi-vendor market, and the large availability of open-source software for networking applications such as Linux [1], Click [2] and the BSD derivatives [3] for the data plane, as well as Xorp [4] and Zebra [5] for the control plane, just to name a few.

Criticisms to software routers are focused on limited performance, software instability, lack of system support, scalability problems, and lack of functionalities. Performance limitations can be compensated by the natural evolution of the PC architecture. Current PC-based routers and switches have the potentiality for switching up to a few Gbit/s of traffic, which is more than enough for a large number of applications. Today, the maturity of open-source software overcomes most problems related to stability and availability of software functionalities. It is therefore important to explore the potentialities and the intrinsic limitations of software routers.

In this paper we focus only on the data plane performance, ignoring all the issues related to management functions and to the control plane. Our aim is to assess the packet forwarding rate of high-end PCs equipped with several Gigabit Ethernet network interface cards (NICs) running at 1 Gbit/s under the Linux operating system.

Provided that, in a PC-based software router, networking functions up to data link layer are performed by NIC hardware, and the IP network layer is implemented as part of the operating system kernel, the paper is organized as follows. Section II gives a quick introduction to the PC architecture, describes the operations and the bandwidth limitations of its key components, and details how a PC can be used as an IP router. Section III, overviews the different implementations of the IP stack available for a Linux-based system. Section IV introduces the experimental setup, describes the tests performed, and comments on the results obtained. Finally, Section V concludes the paper and draws a roadmap for future work.

II. Architecture of a PC-Based Router

A PC comprises three main building blocks: the central processing unit (CPU), random access memory (RAM), and peripherals, glued together by the *chipset*, which provides complex interconnection and control functions.

As sketched in Fig. 1, the CPU communicates with the chipset through the front side bus (FSB). The RAM provides temporary data storage for the CPU as long as the system is on, and can be accessed by the memory controller integrated on the chipset through the memory bus (MB). The NICs are connected to the chipset by the peripheral component interconnect (PCI) shared bus. All interconnections are bidirectional,

0-7803-8924-7/05/$20.00 (C) 2005 IEEE

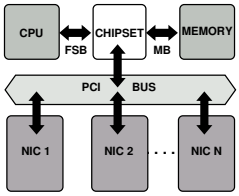

Fig. 1. Key components in a PC-based router

but, unfortunately, use different parallelisms, protocols, and clock speeds, requiring the implementation of translation and adaption functions on the chipset.

State-of-the-art CPUs run at frequencies up to 3.8 GHz. High-end PCs are equipped with chipsets supporting multiple CPUs connected in a symmetric multiprocessing (SMP) architecture. Typical configurations comprise 2, 4, 8 or even 16 identical CPUs.

The front side bus is 64-bit wide and is driven by a *quad-pumped* clock running at either 100, 133, 166, or 200 MHz, allowing for a peak transfer rate ranging from 3.2 to 6.4 Gbyte/s.

The memory bus is usually 64-bit wide and runs at either 100, 133, 166, or 200 MHz with *double-pumped* transfers, providing a peak transfer rate of either 1.6, 2.1, 2.7 or 3.2 Gbyte/s. The corresponding double data rate (DDR) synchronous dynamic RAM (SDRAM) chips are soldered on dual in-line memory modules (DIMM) marketed with the name *PC1600, PC2100, PC2700* and *PC3200* respectively. In high-end PCs the memory bandwidth is further doubled, bringing the bus width to 128 bits, by installing memory banks in pairs. Note that this allows to match the memory bus peak bandwidth to that of the front side bus.

The PCI protocol is designed to efficiently transfer the contents of large blocks of contiguous memory locations between the peripherals and the RAM, without requiring any CPU intervention. As the bus is shared, no more than one device can act as a *bus-master* at any given time; therefore, an *arbiter* is included in the chipset to regulate the access and fairly share the bandwidth among the peripherals. Depending on the PCI protocol version implemented on the chipset and the number of electrical paths connecting the components, the bandwidth available on the bus ranges from about 125 Mbyte/s for PCI 1.0, which operates at 33 MHz with 32-bit parallelism, to 2 Gbyte/s for PCI-X 266, when transferring 64 bits on a double-pumped 133 MHz clock.

Typically, Gigabit Ethernet and Fast Ethernet NICs operate as bus-masters to offload the CPU from performing bulk data transfers between their internal memory and the RAM. Each NIC is connected to one interrupt request (IRQ) line, that is used to notify the CPU of events that need service from the operating system. On the other hand, it is usually possible to switch IRQ generation off altogether, leaving the operating system with the burden of periodically *polling* the NIC hardware and react accordingly.

Summarizing, common PC hardware enables to easily implement a shared-bus, shared-memory router, where NICs receive and transfer packets to the RAM, the CPU routes them to the correct output interface, and NICs fetch packets from the RAM and transmit them on the wire. In such configuration, each packet travels twice through the PCI and the memory bus, effectively halving the bandwidth available for routing traffic. Therefore, a high-end PC equipped with a 1 Gbyte/s 64-bit-wide PCI-X bus running at 133 MHz should be able to feed up to 3-4 Gigabit Ethernet NICs, needing at most 125 Mbyte/s each.

III. IP Network Stacks for Linux

In a PC-based software router, networking functions related to the physical and the data link layer are carried out by NIC hardware, while the IP layer is implemented in software. This section overviews a couple of enhancements to the Linux IP stack and introduces Click [2], an alternative modular IP stack for Linux easily reconfigurable at runtime.

A. Enhancements to the Linux IP Stack

Two long standing issues affecting the networking performance of Unix-like operating systems are receive livelock, described for the first time in [6], and excessive latency in the allocation of packet buffers for the networking subsystem.

Receive livelock affects interrupt driven kernels and originates from a race condition between the NIC hardware-IRQ handler and the network software-IRQ handler. The NIC hardware-IRQ handler just pulls packets out of the NIC reception-ring buffer and moves them to the operating system's backlog queue. The network software-IRQ gets packet from the backlog queue and routes them to the correct output interface, putting them on the NIC transmission-ring buffer. Unfortunately, when the router becomes overloaded with traffic, the software-IRQ handler, which has lower priority than the hardware-IRQ handler, never gets a chance of draining packets from the backlog queue, practically zeroing the forwarding throughput.

The key idea introduced in [6] and implemented in the Linux network stack in [7] with the name new application programming interface (NAPI) easily avoids receive livelock. Fig. 2 sketches the operation of the Linux NAPI network stack: the NIC hardware-IRQ handler is modified so that, when invoked after a packet reception event, it enables polling mode for the originating NIC by switching IRQ generation off and by adding such NIC to the NAPI *polling list*. The networking subsystem then periodically schedules the execution of the `poll` network software-IRQ, which draws packets from the

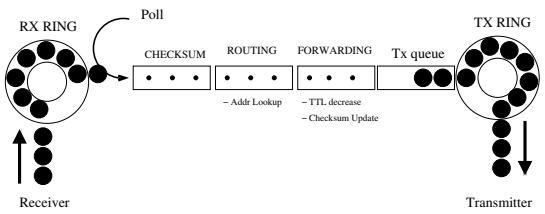

Fig. 2. Operation of the NAPI network stack

reception ring of the NICs on the polling list and routes them to the transmission ring of the correct output NIC. When `poll` finds a NIC reception ring empty, it deletes the NIC from the polling list and re-enables IRQ generation for that NIC. The operation of NAPI is controlled by two parameters: the *quota* Q, which is the maximum number of packets to be removed from any NIC reception ring and the *budget* B, which is the maximum number of packets to be handled in each execution. Of course, B is always greater than Q.

In the standard Linux network stack implementation, buffer management is performed resorting to the operating system general-purpose memory management algorithms, which requires CPU expensive operations. Some time can be saved if the buffer deallocation function is modified so as to store unused packet buffers on a recycling list to speed up subsequent allocations, allowing the device driver to turn to the slower general-purpose memory allocator only when the recycling list is empty. This has been implemented in a patch [8] for 2.6 kernels, referred to as *buffer recycling* patch in the reminder of the paper, which adds buffer recycling functionalities to the `e1000` driver for Intel Gigabit Ethernet NICs.

B. Click

Click is a modular IP stack for Linux running either in kernel space or in user space. Its operations are reconfigurable at runtime just loading a new Click *configuration*, that describes a directed graph connecting a number of different *elements*, as depicted in Fig. 3 for a standard RFC 1812 [9] IP router. The directed graph indicates the flow of the packets in the router, whereas the elements perform different functions such as: verifying the IP header checksum, extracting the IP destination address, routing the packet, decrementing the time-to-live field and fragmenting the datagram if it is larger than the link maximum transfer unit. A configuration can be loaded at runtime with no need of recompiling the operating system kernel or rebooting. See [2], [10] for more detail.

Click can work in both IRQ and polling mode and implements a special-purpose memory management system to allocate/deallocate packet buffers.

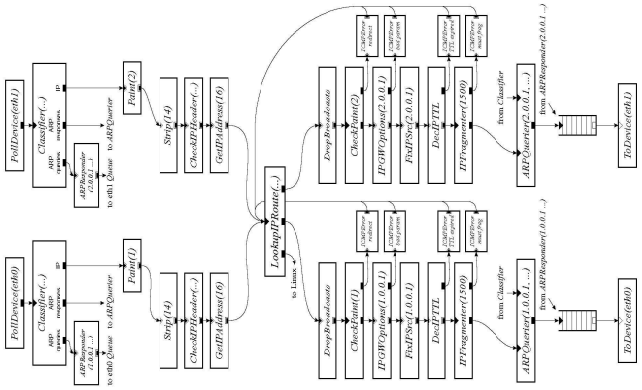

Fig. 3. Example Click configuration for a standard RFC 1812 IP router

IV. Performance Evaluation

The aim of this section is twofold. First, comparing the performance of different network stack configurations in terms of the *saturation forwarding rate* obtained when all router ports are offered the maximum possible load and the system has reached a steady state. Second, devising several tests for gaining a better understanding of the overall system behavior and trying to pinpoint the hardware/software subsystems which most affect the system performance.

Section IV-A introduces the testbed setup, whereas Section IV-B and Section IV-C separately evaluate the maximum reception rate and the maximum transmission rate when forwarding among the different interfaces is disabled. The rationale behind this is to assess to what extent the combination of the operating system kernel, the NIC driver and the NIC hardware can handle either the transmission or the reception of packets when they are performed without doing any other activity. Section IV-D shows performance figures for a router configuration where packets received from a given port are simply transferred to a preassigned output port without even looking at the destination IP address. The aim of this test is to obtain an upper bound on the forwarding rate achievable by the full-blown RFC 1812 router presented in Section IV-E, and to quantify the impact of the per-packet processing and the data touching overhead incurred in the latter case.

A. Testbed Setup

The router tested is based on a high-end PC with a SuperMicro X5DPE-G2 mainboard equipped with one 2.8 GHz Intel Xeon processor and 1 Gbyte of PC2100 DDR RAM consisting of two interleaved banks, so as to bring the memory bus transfer rate to 4.2 Gbyte/s.

For this paper, we build upon the results of the work presented in [11], where we performed a number of transmission/reception tests on several Gigabit Ethernet NICs produced by Intel, 3Com (equipped with a Broadcom chipset), D-Link and SysKonnect, using open-source software generators such as `rude` [12], `udpgen` [13] and `packetgen` [14]. The main conclusion drawn in [11] was that the highest packet generation rate is obtained running `packetgen` on Intel PRO 1000 NICs with the 5.2.52 `e1000` driver. Thus, also in this paper, we present results obtained installing eight Intel PRO 1000 NICs on the router. All driver parameters were left to their default values; NAPI was always enabled, whereas the generation of pause Ethernet frames as well as automatic interrupt rate moderation were disabled in all tests. The NAPI quota Q and the NAPI budget B were left to their default values of 64 and 300 packets. Moreover, since experiments evaluating the packet forwarding rate ran in [11] showed that the buffer recycling patch improves significantly the Linux IP stack performance, all results in this paper were obtained on a patched 2.6.1 kernel, which is the earliest for which the buffer recycling patch is available.

Click was instead installed on a 2.4.21 Linux kernel, the latest for which the `click` kernel module was available. No major changes occurred in the networking code between

kernel version 2.4 and 2.6. The only modification needed to make a fair performance comparison between them is to lower the default 1000 Hz clock interrupt frequency of 2.6 kernels to 100 Hz, which is the default for 2.4 kernels. As far as Click is concerned, all tests were run with NICs in polling mode, whereas the `udpgen` and the `udpcount` configurations bundled in the Click software distribution were used for running transmission and reception experiments.

Notice that, although Gigabit Ethernet NICs offer a raw data rate of 1 Gbit/s, the throughput actually achievable at the IP layer is much lower because of physical and data-link layer overhead. Indeed, the physical layer adds a minimum 12-byte inter-packet gap and precedes each layer-2 frame with an 8-byte preamble, whereas the data-link layer needs an 18-byte overhead and encapsulates a 46- to 1500-byte payload. Consequently, Gigabit Ethernet NICs running at full speed must handle a packet rate ranging between 81 274 and 1 488 095 packets per second [pkt/s] as the payload size decreases from 1500 bytes to 46 bytes.

An Agilent N2X RouterTester [15], equipped with eight Gigabit Ethernet ports, that can transmit and receive Ethernet frames of any size at full rate, was used for generating traffic for reception tests, measuring the traffic generated by the router during transmission tests, as well as sourcing and sinking traffic in routing tests.

Most of the results presented in this section were obtained for minimum-size, i.e., 64-byte, Ethernet frames because they expose the effect of the per-packet processing overhead. However, we also ran a few tests for different frame sizes, to check to what extent the PCI bus and the memory subsystem could withstand the increased bandwidth demand.

We considered both *unidirectional* flows, where each router port, at any given time, only receives or transmits data, as well as *bidirectional* flows, where all NICs send and receive packets at the same time. All the results reported are the average of five runs of the same test, each lasting 30 s.

The IP routing table used in routing tests was minimal and only contained routes to the class-C subnetwork reachable from each port. As a consequence, the number of IP destination addresses to which the RouterTester sent packets on each subnetwork was always less than 255, so that the routing cache never overflowed and the route-lookup overhead was marginal.

All figures contain histogram plots: the scale on the left-hand side is in millions of packets per second [Mpkt/s], whereas the scale on the right-hand side represents the corresponding rate in gigabits per second [Gbit/s], except for Fig. 6(b), where only gigabits per second are reported on the left-hand side.

B. Packet Reception

Considering minimum-size frames, Fig. 4 compares the maximum reception rate achieved by *i*) the standard Linux stack with a NAPI enabled version of `udpcount`, which checks the Ethernet frame protocol-type field and drops the packet (histogram labeled *Linux udpcount*), *ii*) the same configuration but when packets are dropped without checking the Ethernet protocol-type (histogram labeled *Linux no touch*), *iii*) `udpcount` for Click when polling is enabled (histogram labeled *Click udpcount*). The same experiment was repeated receiving traffic at the same time on a number of NICs variable between one and eight, so as to evaluate the scalability in packet reception.

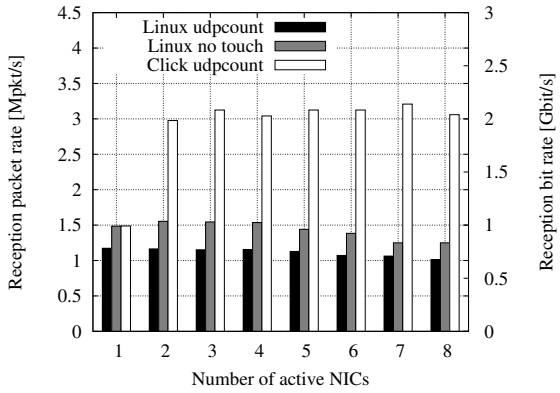

Fig. 4. Comparison of the router reception rate for different IP stacks versus the number of NICs active at the same time

In the *Linux udpcount* configuration, the reception rate slowly decreases from about 1.1 Mpkt/s to 1 Mpkt/s as the number of NICs receiving traffic increases from one to eight. The same holds also for the *Linux no touch* setup, starting from 1 488 095 pkt/s for one NIC, which corresponds to the full 1 Gbit/s line rate for minimum-size Ethernet frames, and ending at about 1.25 Mpkt/s for eight NICs. The difference between the reception rates observed in the two configurations depends on CPU cache misses. Indeed, provided that received packets come from outside the router, trying to read, i.e. *touching*, any of the data they contain, causes a miss in the data cache, stalling the CPU for several clock cycles, while waiting for the requested data to be fetched from the (relatively slower) system RAM.

On the contrary, *Click udpcount* is able to receive all packets sent to one NIC, and, in experiments with two or more NICs active at the same time, receives about 3 Mpkt/s. A quick inspection on the Click source code showed that the programmers introduced some inline assembly instructions to prefetch the packet header before actually accessing it, so as to prevent cache misses from happening.

Notice that, in all cases, the number of packets received in an experiment is equally shared among all the active NICs. The results obtained for one NIC show that, operating carefully, Intel PRO 1000 hardware can receive minimum-size Ethernet frames at full line speed.

C. Packet Transmission

Fig. 5 compares the maximum transmission rate generated by `packetgen` on a baseline Linux stack (histogram labeled *Linux packetgen*), with the maximum transmission rate obtained by `udpgen` for Click (histogram labeled *Click udpgen*).

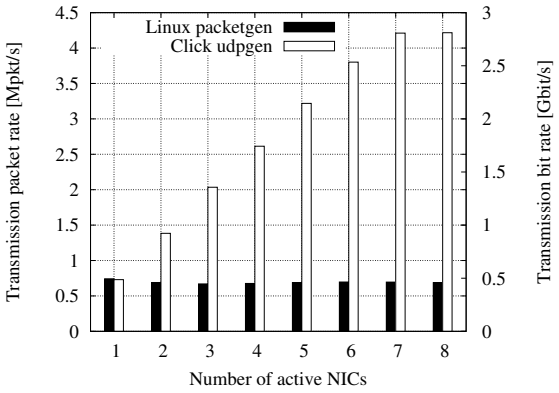

Fig. 5. Comparison of the router transmission rate for different IP stacks versus the number of NICs active at the same time

The aim of the experiment was to assess the impact of CPU loading and of NIC hardware on the packet generation rate. The tests were performed measuring with the RouterTester the maximum transmission rate achieved by the router for 64-byte Ethernet frames when varying the number of NICs generating traffic at the same time between one and eight.

Linux packetgen generates a slightly decreasing packet rate ranging from about 740 kpkt/s to about 690 kpkt/s, as the number of involved NICs increases. The story is pretty different, instead, for *Click udpgen*, which increases the generation rate almost linearly from 730 kpkt/s to 4.2 Mpkt/s as the number of NICs passes from one to seven, and seems to stabilize around 4.2 Mpkt/s for eight NICs. As for packet reception, the number of packets transmitted in each experiment is equally shared among all the active NICs.

The results obtained lead to the astonishing conclusion that, neither Linux nor Click can send minimum-size Ethernet frames at full speed, even when working with one NIC. We were nonetheless able to send a short burst of 64-byte Ethernet frames at full speed, by triggering the transmission after they had been successfully transferred to the NIC internal buffer. This left us with the strong suspect that the transmission limit observed might be originated by an excessive latency of PCI memory-read transfers.

D. Packet Transfer

In this section we evaluate the maximum *transfer rate*, that is, the maximum rate at which packets coming in from a given port can be moved to a preassigned output port, without even looking at their content. The rationale behind this is to prevent cache misses from happening by avoiding any data touching operation, so as to obtain an upper bound on the forwarding rate achievable during normal router operations.

The experiments in this section were performed with a modified Linux kernel in which the IP stack was completely eliminated to implement a fixed port-to-port packet transfer rule. For Click we only wrote and installed a minimal configuration, which instantiates all the interfaces and connects them in pairs, using one packet queue for each traffic flow.

From the packet transmission rates measured in the previous section, it is possible to extrapolate that, very likely, in a unidirectional scenario, the modified router will not be able to transfer more than 740 kpkt/s when running Linux, and more than approximatively 730 kpkt/s times the number input/output interface pairs (with an upper bound of 4 Mpkt/s), when running Click. Taking also into account the reception rates, it is possible to guess that the maximum transfer rate will be limited to no more than 740 kpkt/s under Linux and to no more than 3 Mpkt/s under Click.

Fig. 6(a) shows the transfer rate obtained by Linux and Click, when one, two, three, or four interface pairs are crossed by one, two, three, or four unidirectional traffic flows. Remember that, in each scenario, the router receives packets from half of the interfaces and transfers them to the remaining ones. Each histogram is split into three parts detailing the traffic offered to the router (labeled *Generation rate*), the amount of traffic actually seen by the router operating system (labeled *Reception rate*), and the traffic actually transmitted by output interfaces (labeled *Transmission rate*). For example, the histogram for Linux with one flow shows that the modified router is offered traffic at line rate. However, Linux only receives around 1.4 Mpkt/s because of packet drops in the NIC

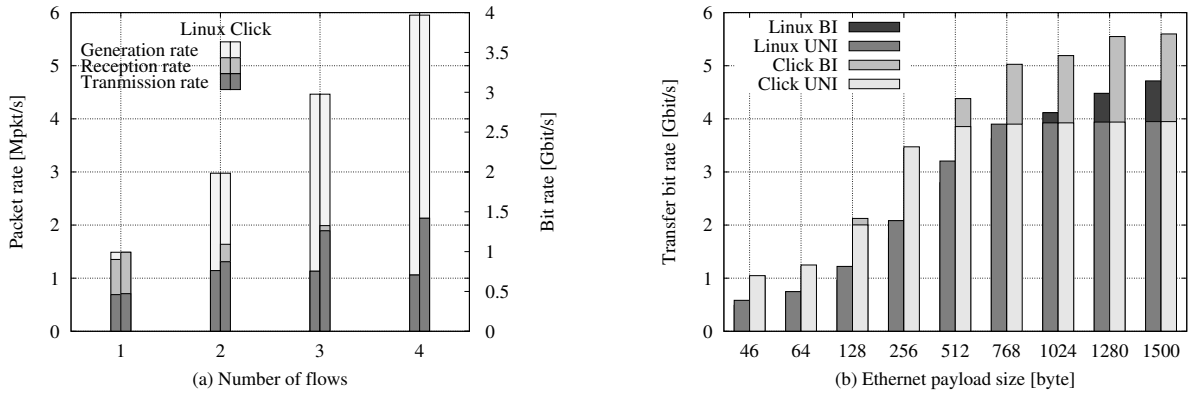

Fig. 6. Comparison of the router transfer rate for different IP stacks: (a) for minimum-size Ethernet frames, when the number of unidirectional traffic flows is varied; (b) for different packet sizes, when the router is crossed by either four unidirectional or four bidirectional traffic flows

reception queue which is not receiving enough service. Finally only 650 kpkt/s get out of the output NIC, the difference being dropped on the Linux transmission queue feeding the NIC. On the other hand, Click can receive all packets generated by the RouterTester, but is bound to the same transfer rate as Linux by the transmission limit of the output NIC.

When using two flows, and therefore two output interfaces, the transfer rate for both stacks becomes approximatively 1.2 Mpkt/s, which is roughly twice the output NIC transmission limit.

Increasing further the number of flows, the packet transfer rate achieved by Linux slightly decreases until about 1 Mpkt/s for four flows, because the system reception limit has been reached. Instead, Click can transfer 1.8 Mpkt/s (i.e., three times as much as the output NIC transmission limit) for three flows, and 2.1 Mpkt/s for four flows, when the system reception limit starts kicking in.

Fig. 6(b) reports the saturation transfer rate in Gbit/s for Linux and Click, when the router is crossed by four traffic flows, either unidirectional (UNI) or bidirectional (BI), and the Ethernet payload size is varied from 46 to 1500 bytes. Notice that the transfer rate achieved for maximum-size (i.e., 1518-byte) Ethernet frames is 5.5 Gbit/s, which corresponds to 11 Gbit/s of traffic flowing across the PCI bus and the system RAM, owing to the shared-bus shared-memory router architecture.

E. Packet Routing

In Fig. 7 we plot the saturation forwarding rate for minimum-size Ethernet frames as the number of flows ranges from one to three. The maximum forwarding rate observed is around 800 kpkt/s, which corresponds to about 500 Mbit/s. The astonishing drop in the performance with respect to the (simpler) packet transfer case depends on the per-packet overhead introduced when looking at the IP destination address and selecting the correct output port.

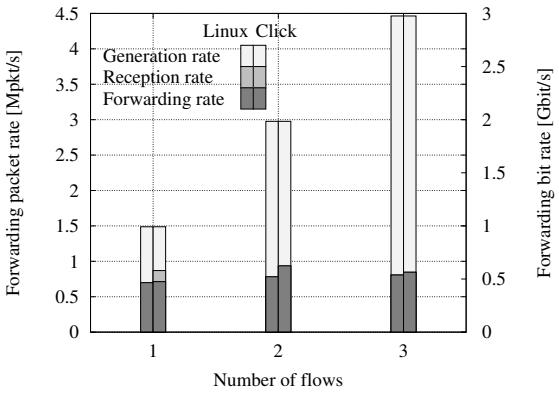

Fig. 7. Comparison of the router forwarding rate for different IP stacks versus the number of flows active at the same time

V. Conclusions and Future Work

In this paper we assessed the feasibility of building a high-performance IP router out of common PC hardware and the Linux open-source operating system. We ran a number of experiments to assess the saturation forwarding rate of both the Linux and the Click IP stacks, completely ignoring all issues related to the control plane.

A software router based on a high-end off-the-shelf PC can transfer up to 750 kpkt/s, corresponding to 500 Mbit/s, when handling 64-byte packets, and 5.5 Gbit/s, when handling 1518-byte packets. Configurations with up to eight ports, each running at 1 Gbit/s, can be easily and inexpensively built at the price of a small decrease in the forwarding rate.

Provided that the major bottleneck in the systems seems to be the per-packet processing overhead introduced by the CPU, we are also profiling the Linux kernel networking code so as to identify the most CPU intensive operations and implement them on custom NICs enhanced with field programmable gate arrays (FPGAs).

Acknowledgment

This work was performed in the framework of EURO [16], a project involving six Italian Universities, partly funded by the Italian Ministry of University, Education, and Research (MIUR).

References

[1] L. Torvalds, “Linux.” [Online]. Available: http://www.linux.org
[2] E. Kohler, R. Morris, B. Chen, and J. Jannotti, “The click modular router,” *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.
[3] “BSD Unix.” [Online]. Available: http://www.bsd.org
[4] M. Handley, O. Hodson, and E. Kohler, “Xorp: An open platform for network research,” in *Proc. of the 1st Workshop on Hot Topics in Networks*, Princeton, NJ, USA, Oct. 28–29, 2002.
[5] GNU, “Zebra.” [Online]. Available: http://www.zebra.org
[6] J. C. Mogul and K. K. Ramakrishnan, “Eliminating receive livelock in an interrupt-driven kernel,” *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 217–252, Aug. 1997.
[7] J. H. Salim, R. Olsson, and A. Kuznetsov, “Beyond softnet,” in *Proc. of the 5th Annual Linux Showcase & Conference (ALS 2001)*, Oakland, CA, USA, Nov. 5–10, 2001.
[8] R. Olsson, “skb recycling patch.” [Online]. Available: ftp://robur.slu.se/pub/Linux/net-development/skb_recycling
[9] F. Baker, “RFC 1812, requirements for IP version 4 routers,” June 1995. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc1812.txt
[10] E. Kohler, “The click modular router,” Ph.D. dissertation, MIT, Cambridge, MA, June 2000.
[11] A. Bianco, J. M. Finochietto, G. Galante, M. Mellia, and F. Neri, “Open-source pc-based software routers: a viable approach to high-performance packet switching,” in *Proc. of the 3rd International Workshop on QoS in Multiservice IP Networks, QoSIP 2005*, Catania, Italy, Feb. 2–4, 2005, pp. 353–366.
[12] J. Laine, “Rude/Crude.” [Online]. Available: http://www.atm.tut.fi/rude
[13] S. Zander, “UDPgen.” [Online]. Available: http://www.fokus.fhg.de/usr/sebastian.zander/private/udpgen
[14] R. Olsson, “Linux kernel packet generator for performance evaluation.” [Online]. Available: ftp://robur.slu.se/pub/Linux/net-development/pktgen-testing
[15] Agilent, “N2X RouterTester 900.” [Online]. Available: http://advanced.comms.agilent.com/n2x
[16] “EURO: University Experiment on Open-Source Routers.” [Online]. Available: http://www.diit.unict.it/euro