Multistage Switching Architectures for Software Routers

Andrea Bianco, Jorge M. Finochietto, Marco Mellia, and Fabio Neri, Politecnico di Torino
Giulio Galante, Istituto Superiore Mario Boella

Abstract

Software routers based on personal computer (PC) architectures are becoming an important alternative to proprietary and expensive network devices. However, software routers suffer from many limitations of the PC architecture, including, among others, limited bus and central processing unit (CPU) bandwidth, high memory access latency, limited scalability in terms of number of network interface cards, and lack of resilience mechanisms. Multistage PC-based architectures can be an interesting alternative since they permit us to i) increase the performance of single-software routers, ii) scale router size, iii) distribute packet-manipulation and control functionality, iv) recover from single-component failures, and v) incrementally upgrade router performance. We propose a specific multistage architecture, exploiting PC-based routers as switching elements, to build a high-speed, large-size, scalable, and reliable software router. A small-scale prototype of the multistage router is currently up and running in our labs, and performance evaluation is under way.

Routers are the key components of modern packet networks and of the Internet in particular. The request for high-performance switching and transmission equipment keeps growing, due to the continuous increase in the diffusion of information and communications technologies (ICT) and new bandwidth-hungry applications and services based on video and imaging. Routers are able to support the performance growth by offering an ever-increasing transmission and switching speed, mostly due to the technological advances of microelectronics.

Contrary to what occurred for personal computers (PC) — where standards were defined, allowing the development of an open, multi-vendor market, at least for the hardware components — the field of networking equipment in general, and of routers in particular, has always been characterized by the development of proprietary architectures. This means incompatible equipment and architectures, especially in terms of configuration and management procedures, as well as the requirement to train network administrators to handle several proprietary architectures or to be limited to a single vendor. This situation yielded commercial practices that are not based on free competition; often, the final cost of equipment is high with respect to performance and equipment complexity. Software routers based on off-the-shelf PC hardware and open-source software are becoming appealing alternatives to proprietary network devices because of the wide availability of multi-vendor hardware, the low cost, and the continuous performance evolution driven by the PC-market economy of scale. Indeed, the PC world benefits from the de-facto standards defined for hardware components that enabled the development of an open market with a wide availability of multi-vendor hardware, low costs offered by the large PC market, wide information available on their architecture, and the large availability of open-source software for networking applications, such as Linux, the Berkeley software distribution (BSD) derivatives, Click [1], Xorp, [2] and Zebra/Quagga [3].

Indeed, despite the limitations of bus bandwidth and central processing unit (CPU) and memory-access speed, current PC-based routers have a traffic-switching capability in the range of a few gigabits per second, which is more than enough for a large number of applications. Moreover, keeping this in perspective, performance limitations are compensated by the natural PC architecture evolution, driven by Moore's law. However, high-end performance cannot be obtained easily today with routers based on a single PC. In addition to performance limitations, several other objections can be raised to PC-based routers; for example, software limitations, scalability problems, lack of advanced functionality, inability to support a large number of network interfaces, as well as the inability to deal with resilience issues to match the performance of carrier-grade devices.

To overcome some of the limitations of software routers based on a single PC, we propose to create a large router exploiting multistage-switching architectures. In addition to presenting the architecture and highlighting its advantages, we also discuss one of the main drawbacks of this proposal, that is, the effort required to coordinate the single-switching elements on the data, control, and management planes so as to make the interconnection of PC behave as a single, large router. Although both the number of PC and internal interfaces increases significantly with respect to the classical stand-alone solution, given the low cost of PC and network interface cards (NIC), the overall architecture may be cost effective when compared with commercial solutions based on the development of application-specific integrated circuits (ASIC). Moreover, no chip re-design efforts are required, because the building blocks are off-the-shelf components, whose performance will increase independently due to the PC-market evolution.

In the framework of the Italian project named Bora-Bora (building open router architectures-based on router aggregation, available at: http://www.tlc-networks.polito.it/borabora) funded by the Italian Ministry of University and Research, we developed in our labs a small-scale prototype with four PC acting as switching elements, with an overall number of 16-Gigabit Ethernet interfaces. Performance analysis obtained using a commercial Agilent router tester is presented; the control and management solution is only described, as its implementation currently is in progress.

Related Work

Multistage switching architectures were previously proposed in different research areas to overcome single-machine limitations [4]. Initially studied in the context of circuit-oriented networks to build large-size telephone switches through simple elementary switching devices, multistage architectures were traditionally used in the design of parallel computer systems and, more recently, considered as a viable means to build large packet-switching architectures.

Indeed, the major router producers have proposed proprietary multistage architectures for their largest routers [5, 6] that follow traditional multistage, telephony-derived switching architectures. In most cases, the routing functionality is distributed among cards installed in different interconnected racks. Such systems target high-end routers, with performance and costs that are not comparable with those of PC-based router architectures. Moreover, high-end routers typically are based on a synchronous behavior, whereas our proposed solution is fully asynchronous. Asynchronous switching relies on sub-systems running on independent clock domains. Therefore, our solution does not require a global clock distribution — a complex task that requires large power consumption and limits router scalability due to the difficulty in distributing synchronization among interface cards and the scheduler — a task even more complex when dealing with multi-rack implementations. Finally, the synchronous behavior often brings about the internal switching of fixed-size data units, which implies segmentation of the variable-size IP packet at inputs and especially, re-assembly procedures at outputs, another complex and power-consuming task.

Similarly to our approach, the IETF is interested in distributed router architecture. The Forwarding and Control Element Separation (ForCES) Working Group [7] aims to propose standards for the exchange of information between the control plane and the forwarding plane, when the control and forwarding elements are either in the range of a small number of hops or even in the same box. A distributed implementation of the control plane on software router architectures was proposed in [8], where unlike in our proposal, no particular switching architecture is addressed, the focus being mainly on control plane issues.

Panama [9], a scalable and extensible router architecture using general-purpose PC with programmable network interfaces as router nodes and a Gigabit Ethernet switch as the router backplane, is a project similar to the one described in this article. However, the proposed multistage architecture is much simpler than the one we propose, as it comprises traditional PC individually acting as independent standard routers. As such, whereas in our architecture coordination among PC is envisioned to make the internal multistage architecture behave externally as a single router, nothing similar is studied in Panama.

To the best of our knowledge, no proposals are available on the use of interconnected PC architectures that explicitly exploit traffic load-balancing to improve performance, scalability, and robustness, as successfully accomplished in Web-server farms. Note that load-balancing at input stages has been shown to be beneficial for scheduling in high-end input-queued switches [10], due to the ability of transforming a non-uniform traffic pattern at input NIC into a uniform traffic pattern at the switching fabric ingress, a feature that we can exploit to overcome single-PC limitations by distributing the computational load among several back-end PC .

Our novel multi-stage architecture exploits classical PC as elementary switching elements to build large routers. The multistage architecture must appear to other routers and to network administrators as a single, large router. Therefore, efforts must be devoted to the management of the distributed solution as discussed later in the article.

The key advantages of our proposed architecture are the ability to:
- Overcome performance limitations of a single-PC-based router by offering multiple, parallel data paths to packets.
- Upgrade router performance by incrementally adding more switching elements or incrementally upgrading each switching element.
- Scale the total number of interfaces the node can host, and as a consequence, the router capacity.
- Automatically recover from faults, that is, reconfiguration can occur in case of any PC/element failure.
- Support a fully asynchronous behavior.
- Provide functional distribution, to overcome single-PC CPU limitations, for example, allowing the offloading of CPU-intensive tasks such as filtering/cryptography to dedicated PC .

Architectural Considerations

Packet-Router Architecture

Referring to Fig. 1, a high-level description of a packet router includes a number of interfaces (mostly bi-directional) toward digital links, called line cards (LC); a control processor (CP), that manages the device and enables its configuration; an internal switching fabric (SF), that enables the information transfer among LC; a data base containing routing information, called routing tables (RT) that can either be centralized in the CP or distributed on the LC. Finally, according to the *store-and-forward* paradigm followed by routers, packets arriving at LC are stored in buffers, either locally in each LC or in a central shared memory.

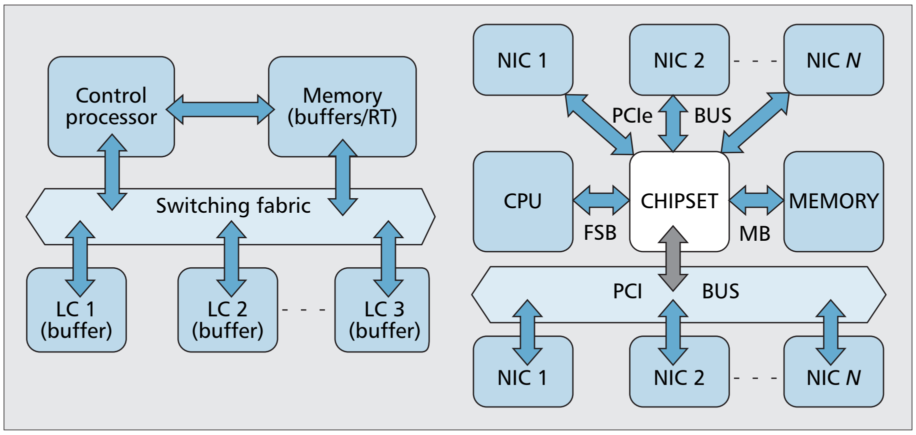
The router architecture is organized in three separate planes, called the *data*, *control*, and *management planes*. The data plane is responsible for all operations involved during packet forwarding; including packet filtering, security check, compression, and so on. The control plane handles all operations required for optimal route calculation, and the management plane enables the network administrator to manage the router and the network. The functionality of the control and management planes, for example, supervision, management, user-access control and security, and so on, are usually implemented by the CP. In particular, the CP interacts (via LC) with other routers using standard protocols to exchange routing information suitable to efficiently build and update the RT.

According to the switching operation defined in the data plane, arriving packets are received at input LC and stored in buffers; then, the routing procedure selects the output LC that enables the packet to reach its final destination. This operation requires the inspection of the information stored in the RT by using the packet IP destination address. The RT inspection can be complex, as a longest-prefix matching (LPM) is required, and it can be delegated to individual LC, which store a local (and possibly partial) copy of the RT. The SF enables the physical transfer of packets between LC. Several different architectures can be adopted in the SF design: bus, shared memory, crossbar, or sophisticated multistage structures. Today, medium- and low-end routers mostly adopt shared buses; whereas simple non-blocking crossbars are generally preferred for high-performance routers. Finally, to solve contention, packets can be stored in buffers before being transmitted.

PC Hardware Architecture

A PC comprises three main building blocks: the CPU, random access memory (RAM), and peripherals, glued together by the *chipset*, which provides advanced 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 and can be accessed by the memory controller integrated on the chipset through the memory bus (MB). Interfaces are connected to the chipset by the peripheral computer interconnect (PCI) shared bus or by a PCI-express (PCIe) dedicated link.



■ Figure 1. *Schematic description of a packet router (left) and a PC architecture (right).*

Today's state-of-the-art CPU run at frequencies up to 3.8 GHz. The front-side bus is 64-bit wide, allowing for a peak transfer rate ranging from 3.2 Gbyte/s to 42.66 Gbyte/s. The memory bus is usually 64-bit wide and provides a peak transfer rate of up to 5.33 Gbyte/s. In high-end PCs, the memory bandwidth can be doubled or quadrupled, bringing the bus width to 128 or 256 bits, by installing memory banks in pairs.

The PCI protocol is designed to efficiently transfer the contents of large blocks of contiguous memory locations between the peripherals and the RAM, using direct memory access (DMA), without requiring any CPU intervention. 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 4 Gbyte/s for PCI-X 266, when transferring 64 bits on a double-pumped 133-MHz clock. Similarly, when a PCIe channel is used, a dedicated serial link between the peripheral and the chipset is used, called a lane. PCIe transfers data at 250 Mbyte/s per lane. With a maximum of 32 lanes, PCIe allows for a total combined transfer rate of 8 Gbyte/s.

When connected by means of either a PCI or PCIe bus, NIC enable a PC to receive and transmit packets, acting as router LC. Therefore, by comparing the router and the PC architecture, it can be seen that common PC hardware enables easy implementation of a shared-bus and shared-memory router. NIC receive and transfer packets directly to the RAM using DMA. The CPU performs packet forwarding by implementing in software the LPM algorithm and then *routes* packets to the correct buffer in RAM, from which NIC fetch packets for transmission over the wire.

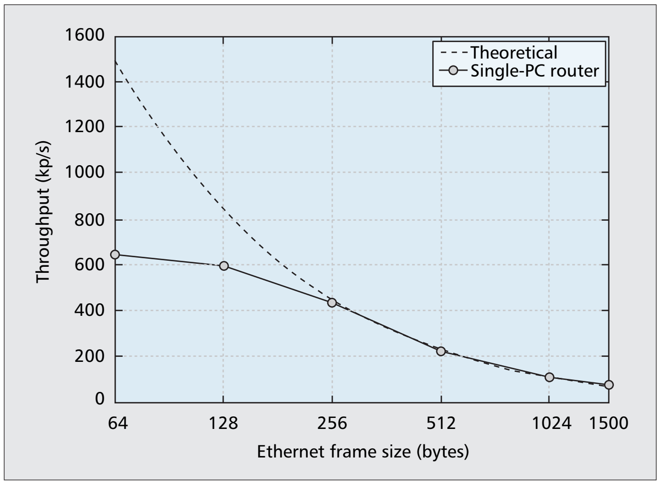
To complete the router design, all data, control, and management plane functionalities must be implemented. Due to the effort devoted to the open software movement by several researchers and software developers, several software distributions, for example, Linux, the BSD derivatives, and Click [1] already offer all the capabilities required to implement data plane forwarding procedures. Similarly, software implementations of the control and management plane exist, for example, Xorp [2] and Zebra/Quagga [3].

Single-PC Performance

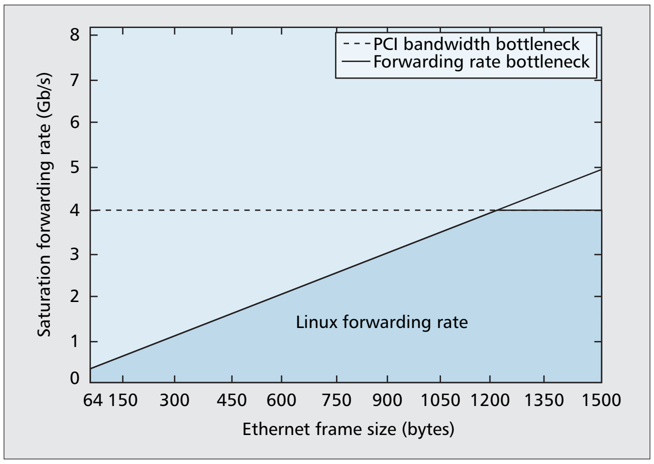
We considered a PC equipped with PCI-X Intel PRO/1000 Gigabit Ethernet line-cards, a single Intel Xeon CPU running at 2.6 GHz, equipped with 1 Gigabyte 128-bit wide, 200-MHz double data rate (DDR) RAM, and PCI-X bus running at 133 MHz, that is, with 8-Gb/s bandwidth as the baseline system. Linux kernel version 2.6.12 was the reference software architecture. State-of-the-art implementation, including optimized drivers (based on the new NAPI NIC/CPU interaction scheme) and optimized memory management (based on buffer recycling), were enabled in our experiments to reach maximum performance [11]. An Agilent N2X router tester 900, equipped with eight Gigabit Ethernet ports that can transmit and receive Ethernet frames of any size at full rate, was used to generate, as well as receive, traffic in routing tests.

Figure 2 reports the throughput in kilopackets per second (kp/s) when a single flow loads the router, that is, packets enter the router from a single Gigabit Ethernet NIC and have to be routed toward a different Gigabit Ethernet NIC. Both the theoretical throughput (dashed curve) and the measured throughput (solid line) are reported. The plot clearly depicts the impact of packet size on performance, showing that a single PC can only reach about 640 kp/s, considering 64-byte-long minimum-size Ethernet frames. This performance figure was verified when considering other NIC, mainboards, and CPU architectures. As pointed out in [11], the limit stems from the high latency faced by the output NIC when requesting (via a DMA operation) a packet from the main memory. Moreover, when more complex operations must be performed by the CPU, for example, imposing access control list (ACL) rules, network address translation (NAT) operations, and so on, the per-packet processing time increases, and the maximum throughput a single PC can sustain is further reduced.

To evaluate the maximum router performance when multi-



■ Figure 2. *Single-PC software-router throughput vs. Ethernet frame size.*



■ Figure 3. *Single-PC software-router saturation forwarding rate vs. Ethernet frame size.*

ple flows simultaneously cross the router, eight 1-Gb/s traffic flows composed by maximum-size Ethernet frames were sent through the router. The resulting maximum throughput was limited to about 4 Gb/s, which corresponds to the bottleneck capacity imposed by the PCI bus that must be crossed twice by each packet to be stored into RAM, processed, and then transmitted.

In summary, the actual capacity of a single-PC software router is limited either by the CPU/memory latency or by the PCI bus capacity that correspond to the operating area highlighted by the shadowed pattern in Fig. 3.

Multistage Architecture

The previously exposed limitations of a single PC drove the design of a multistage architecture. We wished to exploit the optimal cost/performance features of standard PC to design packet-switching devices beyond the limits of a single PC. Given the low cost-per switched bit, we used a non-minimal number of ports and switching elements in the multistage setup, while still being competitive on the overall cost. Several proposals were studied in the literature to implement multistage switching architectures. However, most exploit uni-directional transmission, that is, allowing transfer of information from input to output ports and synchronous behavior, assuming fixed-packet size. Both assumptions fail in our case, as LC have physically bi-directional links, and packets are of variable size. Moreover, according to the PC-based software router concept outlined in the introduction, we wish to use only off-the-shelf networking and relatively cheap PC hardware.

Figure 4 sketches the proposed architecture, where the front NIC of load balancers, on the leftmost part of the figure, act as router I/O cards. The architecture encompasses a first stage of load-balancing switches (L2-balancers) and a back-end stage of IP routers (L3-routers), interconnected by means of standard Ethernet switches. Both L2-balancers and L3-routers are standard PC equipped with several NIC. Packets arriving at the router input ports are:

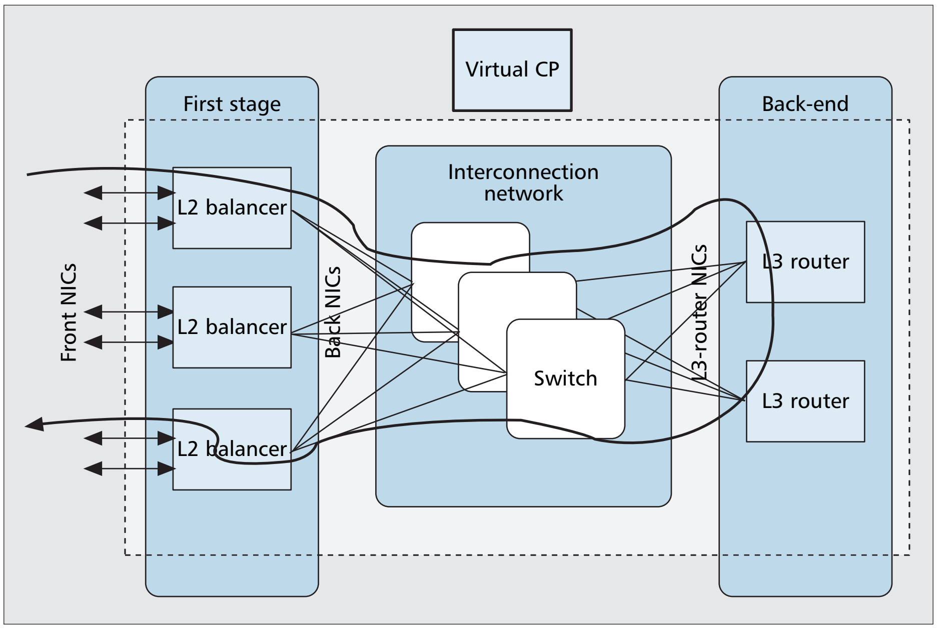
1. Received by an L2-balancer front-NIC, processed by the balancer CPU to perform simple and fast load balancing among back-end routers, and then transmitted by the L2-balancer back-NIC toward the interconnection network.
2. Switched by the interconnection network to the appropriate L3-router NIC.
3. Received by the L3-router and processed by its CPU to perform the required packet operations, then transmitted toward the interconnection network.
4. Switched by the interconnection network to the proper L2-balancer back-NIC.
5. Received by the L2-balancer back-NIC, processed by the balancer CPU to switch the packet toward the appropriate front-NIC, then transmitted toward the next-hop node.

L2-Balancer Operation

The load balancing function at step 1 requires the balancer to forward packets from the front-NIC to the back-NIC, possibly adapting the layer-2 framing formats. Several algorithms can be implemented, from a simple round-robin scheme to more complex algorithms that, for example, guarantee in sequence, routing of packets [12] or balance packets to a particular L3-router based on quality of service (QoS) parameters. Load balancing is obtained simply by setting the destination MAC address of the Ethernet frame, so that the correct L3-router Ethernet NIC is addressed.[1]

In this article, for the sake of simplicity, we consider a simple round-robin scheme and detail a possible implementation in the Linux kernel that we named **DTX** — Direct Transmission. Packets received by the front-NIC are processed directly by the front-NIC driver code, which after changing the destination MAC address according to the load-balancing algorithm, directly calls the back-NIC driver transmission function. All these operations were implemented using Linux kernel primitives, so that packets bypass the TCP/IP Linux protocol stack and are queued directly on back-NIC for transmission. By handling packets at the Ethernet layer, there is no requirement to set up IP-routing tables, and packet processing is minimized. However, advanced packet-manipulation functionality is difficult to obtain by this approach.

To overcome the performance bottleneck that a single NIC



■ Figure 4. *Scheme of the proposed multistage switching architecture.*

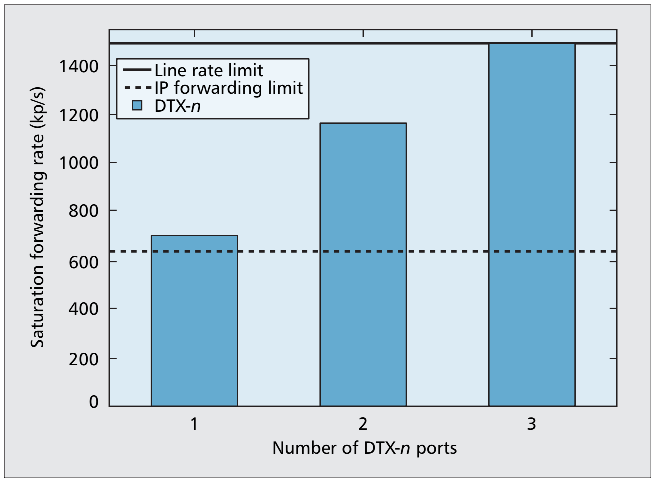


Figure 5. *L2-balancer saturation forwarding rate vs. number of DTX-n ports for minimum-size packets.*

solution faces, *n* back-NIC can be used, in a **DTX-n**— Direct Transmission scheme with *n* back-NIC. In this case, the front-NIC driver performs two round-robin schedules to balance packets on the:
- *n* back-NIC
- L3-router MAC addresses

Finally, considering operations involved at step 5, packets received by L2-balancer back-NIC must be switched according to the destination MAC address to the corresponding front-NIC. The back-NIC driver has a static *forwarding table*, storing next-hop MAC addresses of network devices connected to the front-NIC. When a packet is received by the back-NIC driver, a look-up in the forwarding table is performed to call the correct front-NIC transmission function, therefore causing a direct transmission of the (unmodified) frame toward the next hop. An additional entry is used to deal with broadcast/multicast messages, to correctly copy them to all front-NIC.

Interconnection-Network Operations

Operations involved in steps 2 and 4 are implemented by commercial Ethernet switches, according to the backward learning algorithm and the standard switching behavior. Indeed, the load-balancing among L3-routers is achieved by addressing the corresponding L3-router input-NIC MAC address. Therefore, there is no need to change the normal operation of Ethernet switches. The advantage of using standard Ethernet switches is the reduced costs of these devices, roughly less than $10 per gigabit port.

L3-Router Operations

Operations involved in step 3 are implemented by L3-routers. As standard IP-routing and packet-manipulation operations are used, no changes are required (compared to the standard feature set a single-box router implements). All L3-routers must be correctly configured: IP-routing tables, firewalling, ACL rules, and so on, must be correctly set up.

In summary, both the interconnection architecture and the L3-router stages require only standard functionality; L2-balancers require some minor modifications. Although we propose software implementations using the Linux kernel, we point out that both the load balancing and the switching capabilities that an L2-balancer requires can be implemented in hardware — thus providing better performance — by following an approach similar to the one described in [13]. Indeed, the availability of powerful programmable logic devices permits the extension of the open-software paradigm into the hardware domain. The logic circuitry developed for the field-programmable gate arrays (FPGA) could be made public, reused, and improved by the research community. This open-hardware approach would enable the low-cost implementation of performance-critical functional blocks in hardware. In our labs, we currently are developing an FPGA-based version of the L2-balancer.

Performance Results

We implemented the DTX and DTX-n L2-balancers in a Linux kernel version 2.6.12.We then set up a test bed involving PC with the same hardware configuration as the one used as a baseline reference. An unmanaged 3Com *OfficeConnect* gigabit switch equipped with eight ports was used to build the interconnection network. In the following section we present performance measurements obtained in the resulting test bed, considering both minimum- and maximum-size Ethernet frame scenarios.

L2-balancer Performance

We first tested the performance of a single L2-balancer by loading a front-NIC using the commercial router tester and then directly connecting the back-NIC(s) to the router tester sink(s). Minimum-size Ethernet frames were considered first. Figure 5 reports performance results by comparing the DTX-n solutions versus the number *n* of back-NIC. The theoretical maximum of 1488 kp/s (solid line) and the single-router forwarding limit of 640 kp/s when IP routing is adopted (dotted line) are reported. The DTX-1 solution shows little improvement compared to the single-router IP forwarding limit, confirming that the memory latency is the major system bottleneck. Increasing the number of back-NIC to three guarantees to reach 100 percent throughput in terms of packet rate. These results show that it is possible to achieve line-rate balancing when considering minimum-size frames with a software implementation, but at least three back-NIC are required.

Contrasting the maximum throughput a single L2-balancer offers, we ran a test with flows generating only maximum-size Ethernet frames on a PC with four front-NIC and four back-NIC. The PC was able to sustain about 4 Gb/s, which corresponds to the maximum theoretical throughput when the PCI bus is the bottleneck.

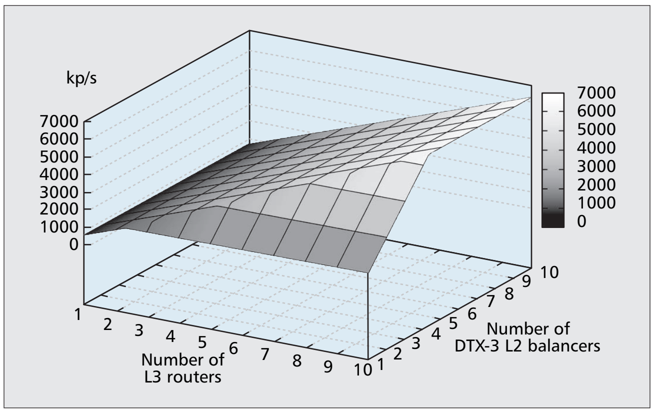
Scaling Performance of the Multistage Router

Performance results of the complete multistage architecture can easily be predicted. Indeed, performance limitations of L3-routers are identical to those of a standard single router, and the Ethernet switches adopted in the interconnection network proved to be capable of sustaining 100 percent throughput even when fully loaded with minimum size packets. In a multistage router formed by N_1 L2-balancers, equipped with at least three back-end NIC each and by N_2 L3-routers, the expected performance in kilopackets per second, considering minimum-size packets is given by:

$$\rho_{min} = \min (N_1 \times 1488, N_2 \times 640)$$

To verify this, we set up multistage architectures with different values of N_1 and N_2 and validated the predicted results. For example, in a test bed with two L2-balancers and two L3-routers, we measured the maximum throughput performance considering minimum size Ethernet frames, obtaining a value of 1280 kp/s, which exactly matched the expected results of doubling the performance of a single-PC router, that is, 2 × 640 kp/s. Figure 6 shows the ρ_{min} vs. the number of L2-balancers (in DTX-3 configuration) and L3-routers. Figure 6 also shows that to reach one Gigabit Ethernet line rate (1488

[1] *In case both front- and back-NIC are Ethernet NIC, a simple re-write operation of the MAC address is required.*



■ Figure 6. *Router performance for minimum-size packets.*

kp/s), it is sufficient to use a single L2-balancer and three L3-routers. Similarly, by considering a setup with four L2-balancers and ten L3-routers, it is possible to route 6 Mp/s, that is, to reach wire speed.

Considering the maximum packet size scenario, each L2-balancer and each L3-router is capable of forwarding four Gigabit Ethernet flows. Therefore, the expected performance of the multistage router in gigabits per second is given by

$$\rho_{\max} = 4 \min(N_1, N_2),$$

which shows that the performance of the multistage architecture increases linearly with the number of switching elements. For example, the same architecture including four L2-balancers and ten L3-routers guarantees a maximum throughput of 16 Gb/s.

Control and Management Plane Implementation Issues

The multistage architecture requires an increase in the total number of ports and switching elements, leading to higher cost (but we use low-cost off-the-shelf components) and to more elements to be controlled and managed. To limit management costs, the interconnected structure must appear to other network routers as a single router, which implies that coordination is required among switching elements. In particular, signaling protocol messages used in the control plane must be correctly received and processed. For example, routing protocol packets must be sent/received only from the proper NIC; TCP connections should be correctly managed by a single entity; router management procedures must hide the details of the internal interconnection to the system/network administrator; and so on. Therefore, the goal is to define a single *virtual CP* architecture, equivalent to a single-router CP that must be capable of transparently managing, configuring, and controlling all switching elements.

Several issues must be addressed to obtain this goal. First, although the considered topology is basically fixed, single PC must be aware of several pieces of information, so automatic bootstrap and topology-management procedures are mandatory. Each switching element must be identified through a unique ID; the number and ID of balancers and the number and ID of L3 routers must be periodically distributed; the number of active network interfaces in each switching element must be declared; and so on. Indeed, this information may change over time due to possible additions, removals, or failures of elements. Second, standard routing protocols, such as Open Shortest Path First Protocol (OSPF), Routing Information Protocol (RIP), Border Gateway Protocol (BGP), Intermediate System to Intermediate System Protocol (IS-IS), and management protocols such as Simple Network Management Protocol (SNMP) must be supported in a coordinated way, as they were designed in the context of a centralized solution in which a single CP is present. Third, routing table computation and distribution must be addressed, so that all L3-balancers share the same configuration. Fourth, control and management plane resilience issues must be handled, so that in case of failure of the current virtual CP, a backup virtual CP is automatically elected. Finally, a decision should be made on whether to support the control and management planes in kernel or in the user space. This issue is more low-level than the previous ones, but it can have important effects in terms of performance and flexibility.

The design of the logical architecture that supports control and management planes could be centralized or distributed. Although the distributed approach is a more elegant solution and may provide some performance advantages due to the load distribution effect, we believe that a centralized approach should be preferred, also taking into account that control and management processing and bandwidth loads rarely constitute a bottleneck. Moreover, the coordination cost in the distributed solution may be fairly large, both in terms of software development and signaling overhead required to support coordination. Finally, for some specific cases, a distributed solution is not viable, because the original design is specifically tied to a centralized approach. For example, all protocols that rely on TCP at the transport layer (e.g., BGP or telnet-like management sessions) can hardly be distributed, since very complex mechanisms are required to correctly deal with TCP connection termination. Note that the centralized approach does not imply running all protocols on a single PC; rather, different protocols may be run on different PCs, but a single entity should exist for each supported protocol.

There are issues to consider about security, fault tolerance, and impact on performance of the control plane. Considering security issues, additional attention must be devoted to avoid intruders from injecting forged control packets that can interfere with internal information distribution. Other than this, no differences apply when comparing the multistage router to a single router. Considering fault management, the distributed architecture offers more flexibility compared to the single-PC architecture, as hot-swapping capability can be implemented as detailed in the following. Finally, impact of control plane management on the switching performance must be carefully investigated. Although the management of each switching element is equivalent to the management of a single router, the overhead due to the multistage architecture management may decrease the performance. However, it should be noted that the PC processing power is continuously increasing, and multicore architectures make parallel processing a cheap and viable solution for this issue. Given this, we selected a solution in which the virtual CP is responsible for terminating all control protocols and for managing all switching elements. To obtain platform independence, we prefer to resort to already available control-plane software, such as Quagga and XORP that run in user space. Indeed, interacting directly with the operating system (Linux or BSD) may provide performance advantages, but it would create a strong dependency on the specific software architecture; and moreover, it would require modifying the kernel architecture. The virtual CP is responsible for receiving/sending all control protocol messages, elaborating the RT, and managing all L2-balancer and L3-router configuration.

Considering the L2-balancer configuration, a simple mechanism to push the list of MAC addresses used by the DTX-n algorithm is required. This can be achieved, for example, by simple messages encapsulated in Ethernet frames that are broadcast to all L2-balancers. These messages can be intercepted at the driver level, so that the list of available L3-router MAC addresses can be updated.

Considering the redistribution of the RT to all L3-routers, instead of distributing the forwarding information base (FIB) routes computed by the virtual CP to the L3-routers, we opt to directly distribute the routing information base (RIB)[2] obtained from each protocol and have each back-end PC calculate its own FIB. One reason is that FIB are operating system (OS)-dependent (being used at the kernel level by the forwarding code), and hence, they may be incompatible in a scenario where the multiple back-end stages may be running different OS. This approach also enables support of FIB partitioning in the future so that each L3-router can be used to route a subset of destinations.

More precisely, on each L3-router, a modified Zebra daemon is running in our setup. A specifically developed additional Zebra module enables communication among remote Zebra daemons. These additional modules are logically connected in a star topology in which the virtual CP Zebra daemon is connected directly to all other modules and enabling the broadcast of the RIB information among all L3-routers.

The management of all PC is obtained through a single console that corresponds to the virtual CP console. It relays commands to and collects information from the specific L3-router module, by explicitly selecting its ID. As such, SNMP management, configuration files management, and static route support for multiple PC is transparently performed by the system administrator as if a single CP were present.

Finally, to support resilience when restarting routing protocol software from a crash, the latest version of the routing information base should be available after the restart. A hot-standby mechanism is established to support a backup of the virtual CP that uses a heartbeat mechanism to detect the crash of the active virtual CP and to activate the correct standby module.

Conclusions

Based on off-the-shelf components and open-source software, we presented performance results of software routers built on PCs. A simple multistage architecture was designed to overcome the intrinsic performance limitations of a single software router. By combining simple layer-2 load-balancing capabilities at the front stage with an array of layer-3 routers at the back stage, the resulting architecture is shown to offer very good scalability properties, for example, enabling the routing of minimum size packets at line rate of gigabit speeds. Moreover, the multistage architecture permits us to increase router performance and scale the number of physical interfaces, and opens the way to a more general distribution of functionality and automatic fault recovery.

Although additional complexity is required to manage the multistage architecture, the benefits of adopting off-the-shelf PC and networking hardware and open-software solutions enables the building of cheap, large-scale routers that can compete with commercial solutions in terms of performance.

Our work shows that software routers can play an important role in the mid- and high-end performance segment of the router market.

Acknowledgments

This work was performed in the framework of the Bora-Bora project funded by the Italian Ministry of University, Education, and Research (MIUR) and developed in the high-quality lab LIPAR at Politecnico di Torino. We wish to thank Robert Birke and Azeem Khan for many discussions and for their valuable comments on control and management plane issues.

[2] *RIB holds all routes from each protocol source. FIB holds the active (best) routes and is used to forward traffic based on an LPM algorithm.*

References

[1] E. Kohler *et al.*, "The Click Modular Router," *ACM Trans. Comp. Sys.*, vol. 18, no. 3, Aug. 2000, pp. 263–97.
[2] M. Handley, O. Hodson, and E. Kohler, "Xorp: An Open Platform for Network Research," *Proc. 1st Wksp. Hot Topics in Networks*, Princeton, NJ, Oct. 28–29, 2002.
[3] GNU, "Quagga," http://www.quagga.net
[4] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Comp. Soc. Press, 1997.
[5] Cisco Systems, "Carrier Routing System," http://www.cisco.com/application/pdf/en/us/guest/products/ps5763/c1031/cdccont 0900aecd800f8118.pdf
[6] Juniper Networks, "Routing Matrix," http://www.juniper.net/solutions/literature/white papers/200089.pdf
[7] IETF, "Forwarding and Control Element Separation (ForCES)," http://www.ietf.org/html.charters/forces-charter.html
[8] H. Hagsand, M. Hidell, and P. Sjodin, "Design and Implementation of a Distributed Router," *Proc. 5th IEEE Int'l. Symp. Sig. Proc. Info. Tech.*, Athens, Greece, Dec. 18–21, 2005.
[9] P. Pradhan and C. Tzi-Cker, "Evaluation of a Programmable Cluster-based IP Router," *Proc. 9th Int'l. Conf. Parallel and Distrib. Sys.*, Taiwan, P.R. China, Dec. 17–20, 2002.
[10] E. Oki *et al.*, "Concurrent Round-Robin-based Dispatching Schemes for Clos-Network Switches," *IEEE/ACM Trans. Net.*, vol. 10, no. 6, 2002, pp. 830–44.
[11] A. Bianco *et al.*, "Click vs. Linux: Two Efficient Open-Source IP Network Stacks for Software Routers," *Proc. IEEE Wksp. High Perf. Switching and Routing*, Hong Kong, P.R. China, May 12–14, 2005, pp. 18–23.
[12] I. Keslassy and N. McKeown, "Maintaining Packet Order in Two-Stage Switches," *Proc. IEEE INFOCOM*, New York, NY, June 23–27, 2002, pp. 1032–42.
[13] A. Bianco *et al.*, "Boosting the Performance of PC-Based Software Routers with FPGA-Enhanced Network Interface Cards," *Proc. IEEE Wksp. High Perf. Switching and Routing*, Poznan, Poland, June 7–9, 2006.

Biographies

Andrea Bianco [M'97] (Andrea.Bianco@polito.it) is an associate professor in the Electronics Department of Politecnico di Torino, Italy. His current research interests are in the fields of protocols and architectures of all-optical networks and switch architectures for high-speed networks. He has co-authored over 100 papers published in international journals and presented at leading international conferences in the area of telecommunication networks. He was Technical Program Co-Chair of High Performance Switching and Routing 2003 and Design of Reliable Communication Networks 2005. He has been a Technical Program Committee member of several conferences, including IEEE INFOCOM, IEEE GLOBECOM, IEEE ICC, HPSR, and Networking.

Jorge M. Finochietto (Jorge.Finochietto@polito.it) is a post-doctoral researcher in the Electronics Department of Politecnico di Torino. His research interests are in the field of switching architectures, scheduling algorithms, and performance evaluation. Since 2002 he has been involved in several European and Italian projects related to optical networks and switching architectures.

Giulio Galante (galante@ismb.it) has been with Istituto Superiore Mario Boella, Torino, since January 2003. His research is mainly focused on the design of vehicular and mesh networks, the performance evaluation of wireless extensions to the TCP/IP protocol suite, and switching architectures based on off-the-shelf hardware and open-source software.

Marco Mellia [M'97] (Marco.Mellia@polito.it) is an assistant professor in the Electronics Department of Politecnico di Torino. His research interests are in the fields of all-optical networks, traffic measurement and modeling, and QoS routing algorithms. He has co-authored over 80 papers published in international journals and presented at leading international conferences, all in the area of telecommunication networks. He has participated in the program committees of several conferences including IEEE INFOCOM, IEEE GLOBECOM, and IEEE ICC.

Fabio Neri [M'98] (Fabio.Neri@polito.it) is a full professor in the Electronics Department of Politecnico di Torino. His research interests are in the fields of performance evaluation of communication networks, high-speed and all-optical networks, packet-switching architectures, discrete event simulation, and queuing theory. He has co-authored over 150 papers published in international journals and presented at leading international conferences. He leads a research group on optical networks at Politecnico di Torino and is the coordinator of the FP6 Network of Excellence e-Photon/ONe on optical networks that involved 40 European institutions. He has been a Technical Program Committee member of several conferences, including IEEE INFOCOM and IEEE GLOBECOM. He was general co-chair of the 2001 IEEE Local and Metropolitan Area Networks Workshop, and of the 2002 and 2007 IFIP Working Conference on Optical Network Design and Modeling. He serves on the Editorial Board of *IEEE/ACM Transactions on Networking* and is Co-Editor-in-Chief of Elsevier's *Optical Switching and Networking Journal*.