PC-based Software Routers: High Performance and Application Service Support

Raffaele Bolla, Roberto Bruschi
DIST, University of Genoa
Via all'Opera Pia 13,
16139, Genoa, Italy
{raffaele.bolla, roberto.bruschi}@unige.it

ABSTRACT
Despite high potential and flexibility in developing new functionalities and mechanisms, as well as the availability of well-established networking SW, common criticism of Software Routers, based on COTS HW elements and open-source SW, is mainly focused on performance, especially for issues concerning the data plane. In this respect, our contribution is aimed at evaluation of architectural bottlenecks limiting the scalability of Software Router performance, and identification of SW and HW enhancements needed to overcome these limitations. Starting from these considerations, we propose a feasible Software Router HW/SW solution able to boost data plane performance while maintaining the flexibility level typical of a SW approach.

Categories and Subject Descriptors
C.2.6 [**Routers**]: Software Router

General Terms
Measurement, Performance, Design, Experimentation.

Keywords
Open Router, Multi-Layer Support, Linux Networking.

1. INTRODUCTION
Internet growth and its widespread adoption as global public multi-service infrastructure have brought about heterogeneous service support and new application-specific needs and requirements. Unfortunately, the current internet infrastructure is composed of interconnection of multi-domain and multi-administrative clouds, which are only concerned with network performance in their own domains, and the responsibility for providing advanced service capabilities (if any) for their direct customers. This is substantially due to the difficulty of establishing business relationships involving multiple administrative domains, and consequently hinders the public internet infrastructure from deployment and support of end-to-end innovative services.

To go beyond this non-evolution of the internet protocol stack in a multi-domain scenario, service architectures over the past few years have been moving from the classical client-server paradigm to Service Specific Overlay Networks (SSONs) [1], with the clear aim of providing end-to-end advanced requirements that are not entirely available at the plain internet network layer. In detail, SSONs are virtual networks deployed at the application layer through logical end-to-end links built on top of existing data transport networks, and are devised to provide capabilities not natively supported by the underlying network (i.e., advanced routing facilities, multicasting, content addressing and caching, media distribution, and related compression/transcoding, etc.).

In this scenario, the development of innovative equipment able to simultaneously act both at the network and SSON layers (and to intelligently share capabilities between them) will be a key point for the evolution and optimization of end-to-end value-added services. In this respect, Software Router (SR) architectures, based on open source software (SW), provide high flexibility and modularity levels, and certainly are considered one of the most promising candidates for developing such advanced multi-layer network nodes. Moreover, when deployed on PC hardware (HW), such SR solutions generally have additional attractive features, such as multi-vendor availability, low-cost, and the ability to continuously update basic parts to ease customization.

In the last few years, growing interest in this kind of equipment has driven some well-known projects (e.g., [2] and [3] among the others) designed to develop complete IP routing platforms on the top of modern open source Operating Systems (OSs) with advanced networking capabilities (e.g., Linux and OpenBSD). Despite the high potential and flexibility in developing new functionalities and mechanisms, as well as the availability of well-established networking SW, common criticisms against SR are mainly focused on the data plane performance. These remarks usually arise from the comparison between SRs, which are based on general purpose HW, and commercial network equipment architectures, where the more intensive and time-critical data plane operations are often offloaded to custom HW components, such as network processors or ASIC/FPGA chips. Today, as shown in [4], the performance gap might be not so large and in any case, is more than justified in many scenarios both by the cost difference and by the flexibility offered by a SW solution.

However, it is currently well-known that PC processor architectures are rapidly adopting the multi-core paradigm, instead of increasing their internal clock or computational capacity, as suggested by Moore's law. Thus, since focusing on squeezing more speed out of a single processor could be a dead end, chip makers are looking at a new class of processor [5], where from 2 to 32 cores divide and conquer the computational load. However, to exploit this increased computational capacity in a SR, we need

PRESTO'08, August 22, 2008, Seattle, Washington, USA.

a suitable SW platform that allows effective packet parallel processing. [6] and [7] show that the forwarding performance levels provided by Linux SMP kernels are usually poor, and generally are lower with respect to the uni-processor (UP) versions. The results reported in [7] underline in greater detail the limitations of the standard Linux kernel ability in effectively parallelizing the packet forwarding process, and also those of exploiting the overall computational capacity of PC multi-processor systems.

Therefore, the aim of this work is to show how an appropriate SR founded on recent PC HW can effectively support advanced application layer services while maintaining a high level of performance in the data plane. In detail, we describe and extensively test a Linux-based networking SW system able to correctly deploy a multi-CPU/core PC architecture. Moreover, we show that this approach can flexibly preserve portions of PC computational power for the application layer, so that the different equipment networking functionalities (namely forwarding, control, and services support) can independently operate with scalable performance. Essentially, the approach is based on a smart CPU/core allocation method (exploiting some new features of recent Intel network boards) that can flexibly distribute the data load on a predefined sub-set of available CPUs/cores. This approach gives the possibility of scaling performance with the CPU/core number and of configuring the SR for a) always having enough-residual power for application services support, and/or b) explicitly reserving specific CPU/cores for the application (and control) layer operations.

The paper is organized as follows. The next section summarizes the Linux approach to the networking data plane. Section 3 describes the main bottlenecks of a PC-based SR, and Section 4 presents the flexible load distribution system and its main aspects. Section 5 reports the results of a set of performance tests demonstrated on conventional and modified SRs, with a specific focus on application layer support and on its interaction with the data plane performance. Finally, the conclusions are summarized in Section 6.

2. THE LINUX DATA PLANE

The packet processing architecture of Linux (and of many other OSs) is generally triggered by HW interrupts: network boards signal the kernel upon packet reception or transmission through HW interrupts. Each HW interrupt (IRQ) is served as soon as possible by a handling routine, which suspends the operations currently being processed by the CPU. The IRQ handlers are designed to be very short, while all time-consuming tasks are performed by the so-called "Software IRQs" (SoftIRQs) afterwards. SoftIRQs differ from HW IRQs mainly in that a SoftIRQ is scheduled for execution by a kernel activity, such as an HW IRQ routine, and has to wait until it is called by the scheduler. SoftIRQs can be interrupted only by HW IRQ routines.

The forwarding process is triggered by an HW IRQ generated from a network device, which signals the reception or the transmission of packets. Then, the corresponding routine performs some fast checks, and schedules the correct SoftIRQ, which is activated by the kernel scheduler as soon as possible. When the SoftIRQ is finally executed, it performs all the packet processing operations by accessing two special RAM regions, called the TxRing and RxRing[1], which are used by the DMA to transfer or receive packets from the network board.

In particular, a SoftIRQ executes two main tasks on the related board, pertaining to packet reception and transmission, respectively. The first task is the de-allocation of already-transmitted packets placed in the TxRing, while the second task includes all the real packet forwarding operations. In fact, the latter task handles the received packets in the RxRing with the NAPI procedure [8], and performs all of the following L2 and L3 delivery operations up to the routing decision and to the backlogging in the TxRing of the selected output interface. Major details on the Linux networking architecture can be found in [4].

With regard to Linux and Multi-CPU support in data plane operations, the 2.6 kernels have reached a high optimization level. The key to these improvements mainly deal with both the optimization and refinement of code locking (to serialize accesses on shared data), and the capacity of binding a process/activity to a specific CPU, called "CPU affinity", thus maintaining very effective use of the processor/core cache.

In addition, the Linux kernel allows the assignment of certain IRQs to specific CPUs/cores through the so-called "IRQ affinity". This has a strong impact on the SR data plane, since it allows control of which CPU/core must process the HW IRQ handling routines (and the following SoftIRQs) for the corresponding network board. Thus, each CPU/Core performs both the de-allocation of packets transmitted by the bound network cards, and the forwarding operations for all packets received by the same boards.

3. ARCHITECTURAL BOTTLENECKS

As shown in [4], when an SR architecture based on a single CPU/core is used, the data plane performance can be substantially limited by only two key factors: the SR computational capacity and the bandwidth/latency of I/O busses. While the latter affects the maximum throughput in terms of bits per second, the former limits the rate of packet headers that can be processed by the CPU.

In this respect, while the PCI-express adoption provides a wide I/O bandwidth, multiprocessor architectures such as SMP and NUMA considerably increase the overall computational capacity, since they both allow parallel processing on multiple CPUs/cores. However, to effectively exploit this kind of HW platform, we need a suitable SW architecture able to parallelize the computational load as much as possible. Typical performance issues, which may sap parallelization gain, are raised when tasks on different CPUs share some data. This generally leads to two issues, namely data accessing serialization and CPU/core cache coherence, which both introduce costly latency times in accessing and in processing shared data.

Starting from the Linux data plane description in the previous section, we can see that TxRings clearly represent data structures shared among the SoftIRQs running on all CPUs/cores involved in the packet forwarding. In fact, the TxRing of a specific board is filled by all CPUs/cores delivering traffic toward that port, while it is only served by the specific CPU bound to its HW IRQ.

[1] In general, there is one Tx and one Rx Ring for each network adapter.

In detail, the SoftIRQ accesses to each TxRing are serialized by a code locking procedure, which is based on the "spin-lock" function. This lock (called "LLTX" lock) guarantees that each TxRing can be read or modified by only one SoftIRQ at a time. When a SoftIRQ gains access to the TxRing, it temporarily stops the following SoftIRQs, which must wait for the TxRing availability. The LLTX lock contention obviously causes computational capacity waste (due to the lock waiting times), especially when several CPUs/cores are involved in the forwarding process.

Moreover, additional performance waste is caused by CPU/core cache management: shared data can be kept in only one processor's cache at a time, otherwise the CPU/core cache may drift out of synchronization (i.e., some processors may work on data that is not up-to-date). Consequently, whenever a CPU/core loads a TxRing to its local cache, all of the other processors also caching it must invalidate their cache copies. Thus, this invalidation is very costly, since shared data can only be cached by one CPU/core at time, but also, it forces the data to be loaded from the RAM every time the processing CPU/core changes. This obviously introduces a non-negligible memory accessing overhead.

The performance analysis in Sub-section 5.1, carried out with both external and internal measurements, clearly shows that LLTX lock and multi-CPU/core cache management cause a heavy SR performance waste that is enough to cancel all the positive effects of using parallel CPUs/cores.

4. MULTI-CPU/CORE ENHANCEMENTS

To go beyond the standard PC-based SW architecture, we need both HW and SW architectural enhancements that allow us to effectively and scalably exploit multi-CPU/core PC systems. This section is divided into three parts. In the first, we introduce some existing HW components that can be used to evolve SR architecture. In the second part, we analyze how the Linux SW architecture may optimally support these HW enhancements for achieving high performance. Finally, in the third part, we provide suggestions for how a SR can effectively include advanced services and network control applications.

4.1 HW evolution

From the HW point of view, Intel has recently proposed two interesting mechanisms that can be effectively adopted to reduce performance issues as much as possible in multi-processor systems. The first one is called "Intel® Advanced Smart Cache" [9], and consists of a mechanism that allows level 2 cache-sharing among all the cores in the same processor[2]. One of the clear advantages offered by this mechanism is the reduction of cache invalidation effects, since, when data (e.g., the TxRing) is shared among a set of cores placed in the same processor, it allows them to avoid unnecessary cache invalidations, and consequently reduces the number of slow memory accesses to RAM.

The second mechanism used to effectively support multi-CPU/core systems pertains to the network board. In fact, Intel PRO 1000 adapters (with MAC chip 82571 and higher) support multiple Tx- and Rx Rings and multiple HW IRQs[3] per network interface [10]. This is particularly useful in achieving higher parallelization levels when different CPUs/cores have to transmit or receive large traffic volumes. This mechanism allows each CPU/core to bind to its own Tx and Rx Rings, reducing the sharing among CPU/cores and avoiding the LLTX lock contention. The traffic multiplexing between the network interface and the Tx/Rx rings is directly managed by the board HW. With regard to multiplexing at traffic reception, this is substantially driven by a hardware-based smart Rx filtering scheme, which can be programmed by the driver (i.e., the driver sends the HW filtering scheme the rules used to associate certain packets to a specific RxRing).

4.2 SW architecture

To exploit the HW mechanisms introduced in the previous sub-section, we need to evolve the SR SW platform with custom developments and a suitable configuration. While the custom SW developments designed to support HW enhancements are already provided in the Intel drivers and were recently included in the official Linux kernel tree (e.g., the code patch supporting multi-queuing adapters written by P.J. Waskiewicz [10] was included in the 2.6.24 kernel), the general SR SW configuration requires a deeper discussion.

In particular, the main objectives of a SR architecture optimized for a multi-CPU/core environment can be summarized by the following :

- to entirely bind all operations carried out in forwarding a packet to a single CPU (minimizing cache invalidations),
- to reduce LLTX lock contention as much as possible,
- to equally distribute the computational load among all the processors/cores in the system.

In this scenario, to achieve these goals, we can play with CPU/core binding to Tx- and RxRings.

CPU/core binding to TxRing: As shown in Section 3, TxRings represent the real weak point for standard SR architecture. They are shared data structures, causing both a high rate of cache invalidations among CPUs/cores and heavy LLTX lock contentions. The optimal solution is clearly to bind each CPU/core to a different TxRing on each output device. If an adapter does not include enough TxRings, the same TxRing can be assigned to a set of cores sharing the L2 cache; even if this sub-optimal solution does not mitigate LLTX lock contention, the core cache sharing minimizes the rate of cache invalidations, and consequently, of slow RAM accesses.

CPU/core binding to RxRing: Each CPU/core is characterized by a certain computational capacity, which limits the maximum processing throughput of packet headers. Observing the performance measurement in Section 5, we can note that a single Xeon core allows the processing of about 1 Mpkt/s, which is too low a value to manage a gigabit Ethernet interface at line speed (1.488 Mpkt/s with 64B sized frames), while it is enough to manage more than six Fast Ethernet interfaces (148.8 kpkt/s with 64B sized frames). Starting with these considerations, a suitable approach would be to assign enough processing capacity to each network interface to achieve its maximum theoretical speed. Thus, we can bind a single CPU/core to a set of "slow-speed" interfaces, while we have to bind multiple cores to the same Rx board in the presence of high speed interfaces (1 or 10 Gbps). The first case corresponds to the standard Linux behavior, since we do not need multiple RxRings per interface. Alternatively, for high-speed network adapters, we have to enable multiple RxRings, binding each one to a different CPU/core.

[2] Note that in standard multi-core processors, each core has its own independent level 2 cache.

[3] A large number of "IRQ channels" is possible by using MSI-X.

Thus, as shown in Fig. 1, a SR architecture fully optimized for multi-CPUs/cores environment must provide:

- a number of CPUs/cores per Rx port that guarantees enough computational capacity to process incoming traffic
- a number of TxRings per interface equal to the number of CPUs/cores involved in traffic forwarding
- multiple RxRings per high-speed interface (ideally equal to the number of CPUs/cores need to achieve the maximum theoretical packet rate).

Note that when multiple RxRings are used, the smart Rx filtering scheme embedded in the high-speed adapter has to balance the traffic load among all the enabled RxRings with a suitable mechanism. In particular, to avoid out-of-order packet delivery, it must assign to the same RxRing all packets of a certain flow, while at same time equally distributing the traffic load on all RxRings.

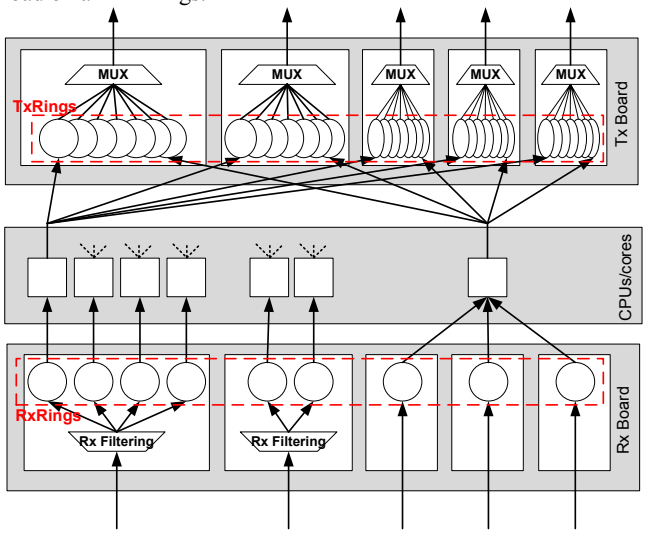

Fig. 1. Overview of the enhanced SR architecture.

4.3 Application service support

To develop a multi-layer node, the SR has to include SW modules in its architecture to support SSON functionalities. These advanced functionalities, as well as classical applicative services and network control applications (e.g., [3]) are usually realized in the user-space as applications. The interaction among user-space and the lower layers is often effectively performed through sockets, which flexibly allow access to the network protocol stack, and also through other well-known APIs provided by the OS.

Notwithstanding, PC-based architectures are the most obvious environment used to support applications, and the coexistence between these activities and Linux data plane operations may be critical. This arises from the packet forwarding process, which is fully realized as SoftIRQ activity in kernel-space, and consequently behaves as a higher priority task with respect to other activities (e.g., SR services and applications in user-space). Therefore, as shown in Sub-section 5.3, when all of the CPUs/cores in the system perform intensive forwarding operations, SR services and applications may not receive enough computational capacity for correct operation. To address these considerations, a viable and safe solution consists of allocating a suitable number of CPUs/cores to the SR services and applications. In the 2.6 kernels, this can be fulfilled through non-binding of these CPUs/cores to any network board; the kernel scheduler will automatically place services and applications on free CPUs/cores.

5. PERFORMANCE EVALUATION

This section reports a set of benchmark results obtained by analyzing and evaluating the performance of the standard and enhanced SR architectures. In greater detail, the studied architectures are based on a custom 2.6.24 Linux kernel, which includes a set of "ad-hoc" optimizations, already studied and introduced in our previous works [4].

During benchmarking activities, we used both external and internal measurement tools. To measure SR throughput and latency performance, we adopted professional equipment, namely an Agilent N2X router tester, while for internal measures we utilized Oprofile [11].

This section is organized in three different parts, which report the performance of standard SR architecture and bottleneck characterization, the performance provided by the enhanced SR, and a benchmarking session used to evaluate the SR support for multi-layer services and applications.

5.1 Standard SR architecture

This benchmarking session consisted of a forwarding performance evaluation of standard SR architecture without any HW and SW enhancements. We used two different SMP kernel setups in a very simple environment: we consider only traffic flow crossing the SR among two gigabit Ethernet interfaces. The packet size is fixed to 64B, since computational capacity is one of the major performance bottlenecks. The kernel setups used are: 1) *SMP 1-CPU*: in this kernel configuration, both the crossed network adapters are bound to the same CPU; 2) *SMP 2-CPU*: in such a case, the two network adapters are assigned to different CPUs. While the former (SMP 1-CPU) attempts to maximize the CPU affinity of the forwarding process, the latter (SMP 2-CPU) performs load balancing among all CPUs. The performance results obtained with a uni-processor (UP) kernel are used for comparison. Fig. 2 shows the throughput and latency values obtained with all studied kernel setups. While the UP kernel forwards at about the full gigabit speed, the SMP kernel versions show lower performance values: the SMP 1-CPU achieves a maximum forwarding rate equal to about 1020 kPkt/s, while the SMP 2-CPU performs to about 40% of gigabit speed. We can justify the performance gap among SMP and UP kernels with the additional complexity overhead required with SMP kernels (e.g., needed to manage kernel spinlock, etc.). However, there is a clear performance gap between the SMP setups as well: the performed tests show that load balancing of the forwarding processes among different CPUs generally leads to a substantial performance decay. Fig. 2 shows the number of cache misses (that obviously depend on the cache invalidation rate) and the average spinlock waiting time (where a significant contribution is provided by the LLTX lock), both obtained with Oprofile and for all three SW setups. Observing these figures, we can highlight how the number of cache misses and spinlock waiting times are notably larger in the SMP 2-CPU setups. This confirms what was introduced in Section 3 regarding the cache invalidation bottleneck: when packets cross the SR among interfaces bound to different CPUs, they cause a large number of cache invalidations and a consequent memory access slowing down. For spinlock waiting times, we can see that in the SMP 2-CPU they double their average value with respect to the SMP 1-CPU, in the presence of only two concurrent CPUs at the LLTX lock.

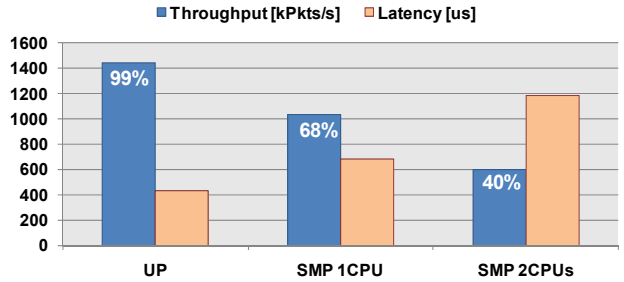

Fig. 2. Maximum throughput and latency values with L2 64 B sized packets.

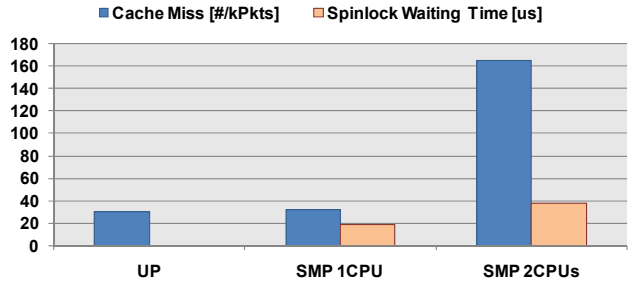

Fig. 3. Number of cache misses and average spinlock waiting times per forwarded kPkts.

5.2 Enhanced SR architecture

This sub-section reports results from benchmarking sessions carried out to analyze the enhanced SR architecture. The SR HW is based on a dual 64 Xeon system, where each Xeon provides four cores, and on an Intel Gigabit adapter based on the 82575 chipset, which provides four Tx- and Rx-Rings per port.

We performed four different benchmarking sessions according to an increasing number of cores and gigabit interfaces involved in the forwarding process: for every two cores, we added a Rx and Tx gigabit interface. In particular, for each Rx gigabit network interface, we bound two cores, each one to a different RxRing. Where possible, a TxRing is associated with a single core on every Tx network interface. However, since the network board only supports up to four TxRings per interface, when we use more than four cores, we bound some TxRings to two cores. Thus, in the benchmarking sessions with two and four cores, we have a fully optimized SR architecture (i.e., each Tx and Rx Ring is only bound to a single core), while in the sessions with six and eight cores, we can only realize a sub-optimal SR configuration, since some TxRings are shared among two cores. For the test traffic, we used a partially meshed matrix composed of unidirectional flows that cross the SR from each Rx interface to each Tx. The load on each Rx port saturates the gigabit link with L2 64B sized packets, and is equally distributed among the traffic flows. Note that we only utilized L2 64B sized packets, since we want to focus on how the enhanced SR architecture can handle the computational bottleneck (in terms of processed packet headers per second) with respect to the standard architecture.

Fig. 4 shows the maximum throughput in terms of kPkts/s, for both the enhanced and standard SR architectures, obtained in all the setups. From this figure, we note that the enhanced SR provides very high performance with respect to the standard one. In detail, while the standard SR holds about the same packet forwarding throughput (i.e., about 500-600 kPkts/s) independent from the number of active CPUs and network interfaces, the enhanced architecture seems to offer a quasi-linear performance scalability. The setups with two and four cores (i.e., the fully optimal configurations) achieve the maximum theoretical rate on one and two gigabit Ethernet interfaces, respectively. The sup-optimal configurations (with six and eight cores), where we have three and four couples of Rx and Tx gigabit interfaces, show a performance decay; with six cores we achieve a maximum throughput equal to about 4.4 MPkts/s, while the SR forwards up to 5.9 MPkts/s in the setup with eight cores, which corresponds to about the 70% of the maximum theoretical throughput. Thus, while optimal SR configurations can achieve the full rate speeds, even the sub-optimal configurations guarantee a high performance level.

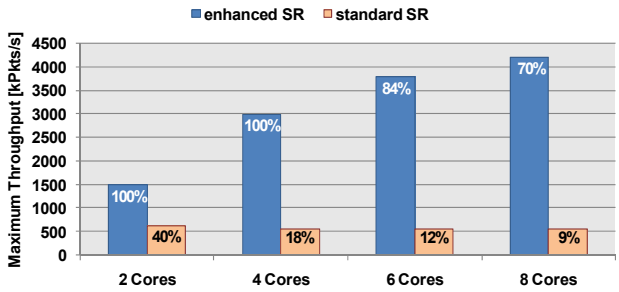

Fig. 4. Maximum throughput for both the enhanced and the standard RS with according to 2, 3 and 4 core setups.

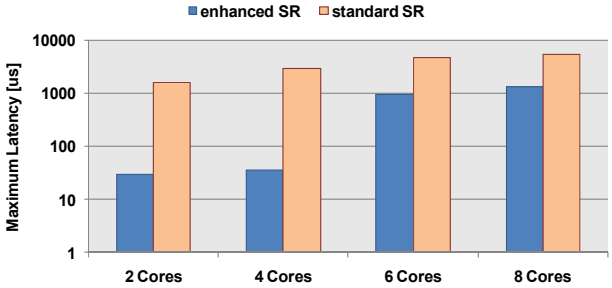

Fig. 5. Maximum latency values for both enhanced and standard RS with according to 2, 3 and 4 core setups.

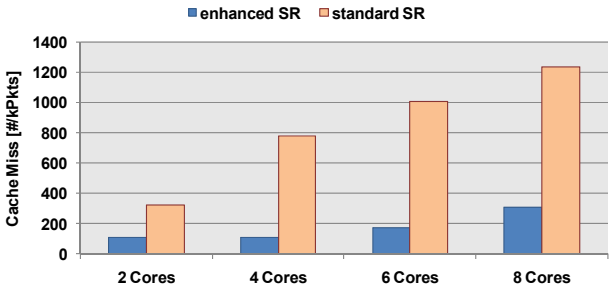

Fig. 6. Cache miss rate for both enhanced and standard RS with according to 2, 3 and 4 core setups.

Fig. 5 reports the maximum latency times measured in all performed setups for the standard and enhanced architectures. The results are consistent with respect to those related to the maximum throughput in Fig. 4; the enhanced SR guarantees lower latency times than the standard one. This is a clear indication that TxRing un-sharing (especially in the optimal configurations with two and four cores) and Rx balancing among multiple cores help to lower the average processing time of forwarded packets. Finally, the results concerning the cache miss per forwarded KPkt, are reported in Fig. 6. As highlighted in the figure, the enhanced architecture limits the number of cache invalidations and also cache misses. In the standard SR architecture, the number of cache misses seems to increase linearly according to the number of cores involved in the forwarding process.

5.3 Multi-layer service support

As sketched in Sub-section 4.3, the coexistence of applicative services and the Linux data plane may be critical, since kernel level operations have a higher priority than user-space activities. To this purpose, we decided to perform several internal measurements, aimed at estimation and quantification of the impact of the forwarding process impacts on performance of service applications.

Fig. 7 shows the SW profiling of a SR (obtained with Oprofile), where the data plane and the applicative services are forced to work on the same CPU. The tests have been carried out against different loads (up to 1 Gbps with 64B sized packets). Among several interesting observations related to SR data-plane internal dynamics (for major details see [4]), Fig. 7 clearly outlines that the applicative services, namely "User-Space", rapidly decrease their computational occupancy as the load increases and kernel activities become more intensive. In detail, when the traffic load exceeds 300 Mbps, the applicative services exploit less than the 2% of the overall CPU capacity. With the aim of verifying the solution proposed in Sub-section 4.3 (to guarantee at least a free CPU to applicative services), we conducted further SW profiling sessions, similar to the previous one, but with an additional CPU not involved in data-plane operations. In such a case, as we can observe in Fig. 8, even if the service and data-plane processes initially share the same core (the worst case condition), when traffic load reaches 10%, the Linux kernel scheduler decides to move all the service applications to the free core, which can consequently exploits 70-75% of CPU time.

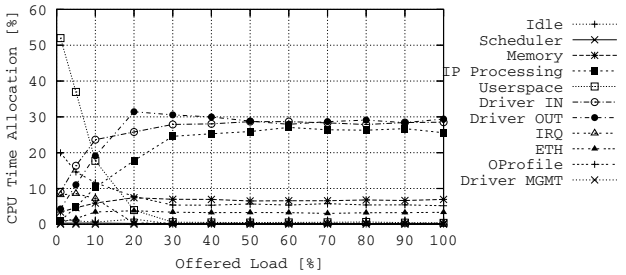

Fig. 7. Single core processing data plane operations and applicative services: CPU utilization against the offered load.

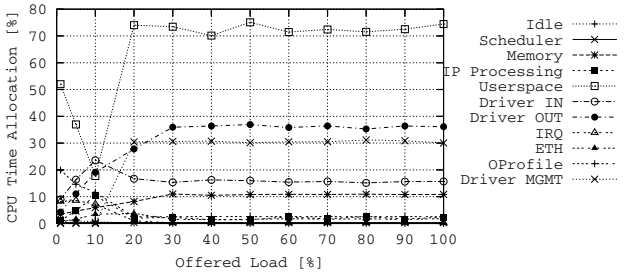

Fig. 8. Multiple cores processing data plane operations and applicative services: CPU utilization against the offered load.

Conclusion

In this contribution, we reported an in-depth study of a PC-based SR architecture, focusing on the SW/HW enhancements that can be used to achieve high performance and to simultaneously support applicative services.

We described and extensively tested a Linux-based networking SW system, able to correctly deploy multi-CPU/core PC architecture. The approach, based on a smart CPU/core allocation to multiple network board Rx and Tx buffers, gives the possibility of scaling performance and effectively supporting a certain number of high speed devices (we achieved a maximum throughput equal to about 4MPkts/s).

Moreover, this approach allows us to flexibly preserve portions of the PC computational power for the application layer, so that different equipment networking functionalities can independently operate with scalable performance.

6. ACKNOWLEDGMENTS

Thanks to P.J. Waskiewicz for his kindly support.

7. REFERENCES

[1] Duan, Z., Zhang, Z. and Hou, Y. T., 2002. Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning. In Proc. of the 10th IEEE Int. Conf. on Network Protocols (ICNP'02), Paris, France, 1-10.

[2] Kohler, E., Morris, R., Chen, B., Jannotti, J. and Kaashoek, M. F., 2000. The Click modular router. ACM Trans. on Computer Systems 18, 3 (Aug. 2000), 263-297.

[3] Handley, M., Hodson, O. and Kohler, E., 2003. XORP: an open platform for network research. ACM SIGCOMM Computer Communication Review 33, 1 (Jan 2003), 53-57.

[4] Bolla, R. and Bruschi, R., 2007. Linux Software Router: Data Plane Optimization and Performance Evaluation Journal of Networks (JNW) 2, 3, Academy Publisher, 6-11.

[5] Geer, D., 2005. Chip makers turn to multicore processors. Computer 38, 5 (2005), 11-13.

[6] Bolla, R. and Bruschi, R., 2008. An Effective Forwarding Architecture for SMP Linux Routers. Proc. of the 4th Int. Telecom. Networking Workshop on QoS in Multiservice IP Networks (QoS-IP 2008), Venice, Italy, 210-216.

[7] Bolla, R. and Bruschi, R., 2006. IP forwarding Performance Analysis in presence of Control Plane Functionalities in a PC-based Open Router. In Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, Springer, Norwell, MA, 143-158.

[8] Salim, J. H., Olsson, R., and Kuznetsov, A., 2001. Beyond Softnet. Proc. of the 5th annual Linux Showcase & Conf., Oakland, CA, USA.

[9] Doweck, J., 2006. Intel Smart Memory Access: Minimizing Latency on Intel Core Microarchitecture. Technology Intel Magazine, Sept. 2006, 1-7.

[10] Yi, Z. and Waskiewicz, P.J., 2007. Enabling Linux Network Support of Hardware Multiqueue Devices. Proc. of 2007 Linux Symposium, Ottawa, Canada, June 2007, 305-310.

[11] Oprofile, http://oprofile.sourceforge.net/.