The IP Lookup Mechanism in a Linux Software Router: Performance Evaluation and Optimizations

Raffaele Bolla, Roberto Bruschi
DIST - Department of Communications, Computer and Systems Science
University of Genoa
Via Opera Pia 13, 16145 Genova, Italy
{raffaele.bolla, roberto.bruschi}@unige.it

***Abstract*—In the last years, networking device architectures based on Open Source Software, like Linux, have aroused lively interest from both Scientific and Industrial Communities. The key to this success can probably be found in the flexibility and fast development time of the Software approach, and the reliability level guaranteed by large communities of users and developers. In this contest, our aim is to customize, to analyze and to optimize a Linux based architecture for an exclusive networking use. In particular, the objective of this work is to study, to evaluate the performance and to optimize the IP lookup mechanism included in the Linux kernel. With this aim we have released a set of kernel patches to enhance the performance of the system by optimizing the IP lookup mechanism. We present a complete set of benchmarking results with both internal and external measurements.**

Keywords-Open Router; IP lookup; Linux Router.

I. INTRODUCTION

In the last years, network equipment architectures based on Open Source software have received a great interest from the Scientific and Industrial Communities. Some well known projects (i.e., the Linux kernel itself, the Click Modular Router [1], Zebra [2] and Xorp [3]) have been successfully activated and carried out with the main aim of developing a complete IPv4 routing platform based on Open Source Software (SW) and COTS Hardware (HW). In the following, we refer to this platform as Open Router (OR). Some interesting works regarding ORs can be found in the scientific literature. Among the others, in [4], [5] and [6], Bianco et al. report a detailed performance evaluation. [7] and [8] propose a router architecture based on PC cluster, while [9] reports some performance results (in packet transmission and reception) obtained with a PC Linux-based testbed. In our previous works, carried out inside the BORABORA project [10] [11] [12], we have focused our attention on testing, optimizing and evaluating the performance of the basic packet forwarding functionalities. To this purpose, besides classical external (throughput and latency) measurements, we have adopted also profiling tools to analyze in depth and to optimize the internal behaviour of a Linux based OR. In [13], we have proposed a survey on how the presence of control plane functionalities in Software Router can impact on the forwarding performance. The previous work concerning Linux Software Router shows that large interesting areas still require a deeper investigation: the identification of the most appropriate HW structures, the comparison of different SW solutions, the identification of the best SW configurations with an indication of the most significant parameters, the accurate characterization of the open node performance, the identification of SW and HW bottlenecks and the effectiveness and influence of advanced functionalities (e.g. firewalling, QoS, etc.) are only some of the open issues in this field of research. Starting from a optimized Linux OR architecture, the objective of this work is to focus our attention on the IP lookup mechanism included in the last generation Linux kernel. With this aim, we present a detailed analysis of the Linux IP lookup mechanism, its performance evaluation, and some kernel optimizations to enhance the whole system behaviour. In particular, we focus our attention both on the RT cache performance and on the performance comparison of the two IP lookup mechanism included in the Linux kernel: the Radix and the LC-trie. In [14] is reported an interesting and detailed analysis of LC-trie mechanism, its Linux implementation is taken into account and tested, but, contrary to the performance evaluation reported in this work, the IP lookup has been tested as a standalone module (i.e., without the complete Linux forwarding chain).

The paper is organized as in the following. Section II includes the SW OR architecture description, while in Section III we report a detailed description of the proposed kernel patches/enhancements. In Section III and Section IV, the benchmarking scenario and the performance results are reported. The conclusions are in Section V.

II. THE LINUX OR ARCHITECTURE

As outlined in [12], while all the forwarding functions are realized inside the Linux kernel, the large part of the control and monitoring operations (the signalling protocols like, for example, routing protocols, control protocols, …) are daemons/applications running in user mode. Thus, we have to outline that, unlike most of the high-end commercial network equipments, the forwarding functionalities and the control ones have to share the CPUs in the system. [13] reports a detailed description of how the resource sharing between the control plane and the forwarding process can have effect on the overall performance in different OR configurations (e.g., SMP kernel, single processor kernel, etc.). The critical element for the IP forwarding is the kernel where all the link, network and transport layer operations are realized. During the last years, the networking support integrated in the Linux kernel has experienced many structural and refining developments. For these reasons, we have chosen to use a last generation Linux kernel, more in particular a 2.6 version. In the following sub-sections, we report a short overview of the Data Plane architecture and a description of the IP lookup mechanism included in the Linux kernel. Since our main aim is to study

and to optimize the OR performance focusing on the IP lookup mechanism, in the following two sub-Sections, we describe a description of the whole Linux forwarding architecture, and of the IP lookup mechanism, respectively.

A. The Linux Data Plane Architecture

Since the older kernel versions, the Linux networking architecture is fundamentally based on an interrupt mechanism: network boards signal the kernel upon packet reception or transmission, through HW interrupts. Each HW interrupt is served as soon as possible by a handling routine, which suspends the operations currently processed by the CPU. Until completed, the runtime cannot be interrupted by anything, even by other interrupt handlers. To make reactive the system, the interrupt handlers are designed to be very short, while all the time consuming task are performed by the so called "Software Interrupts" (SoftIRQ) in a second time. This is the well known "top half – bottom half" IRQ routine division implemented in the Linux kernel [15]. SoftIRQs are actually a form of kernel activity that can be scheduled for later execution rather than real interrupts. They differ from HW IRQs mainly in that a SoftIRQ is scheduled for execution by an activity of the kernel, like for example a HW IRQ routine, and has to wait until it is called by the scheduler. SoftIRQs can be interrupted only by HW IRQ routines. The "NET_TX_SOFTIRQ" and the "NET_RX_SOFTIRQ" are two of the most important SortIRQs in the Linux kernel and the backbone of the whole networking architecture, since they are designed to manage the packet transmission and reception operations, respectively. In details, the forwarding process is triggered by a HW IRQ generated from a network device, which signals the reception or the transmission of packets. Then the corresponding routine makes 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 forwarding operations. About the packet forwarding process, it is fundamentally composed by a chain of three different modules: a "reception API" that handles the packet reception (NAPI), a module that carries out the IP layer elaboration and, finally, a "transmission API" that manages the forwarding operations to the egress network interfaces. In particular, the reception and the transmission APIs are the lowest level modules, and are composed by both HW IRQ routines and SoftIRQs. They work by managing the network interfaces and performing some layer 2 functionalities. Analyzing more in details, the modules that compose the forwarding chain, the NAPI [16] was introduced in the 2.4.27 kernel version, and it has been explicitly created to increase the reception process scalability. It handles network interface requests with a interrupt moderation mechanism, which allows to adaptively switch from a classical interrupt management of the network interfaces to a polling one. In particular, this is done by inserting, during the HW IRQ routine, the identifier of the board generating the IRQ to a special list, called "poll list" and scheduling a reception SoftIRQ, and disabling the HW IRQs for that device. When the SoftIRQ is activated, the kernel polls all the devices, whose identifier is included in the poll list, and a maximum of quota packets are served per device. If the board buffer (RxRing) is emptied, then the identifier is removed from the poll list and its HW IRQs re-enabled, otherwise its HW IRQ are left disabled, the identifier kept in the poll list and a further SoftIRQ scheduled. While this mechanism behaves like a pure interrupt mechanism in presence of low ingress rate (i.e., we have more or less a HW IRQ per packet), when traffic raises, the probability to empty the RxRing, and so to re-enable HW IRQs, decreases more and more, and the NAPI starts working like a polling mechanism. For each packet, received during the NAPI processing, a descriptor, called *skbuff*, is immediately allocated and used for all the layer 2 and 3 operations. A packet is elaborated in the same NET_RX SoftIRQ, till it is enqueued in an egress device buffer, called Qdisc. Each time a NET_TX SoftIRQ is activated or a new packet is enqueued, the Qdisc buffer is served. When a packet is dequeued from the Qdisc buffer, it is placed on the Tx Ring of the egress device. After the board transmits one or more packets successfully, it generates a HW IRQ, whose routine schedules a NET_TX SoftIRQ. During a NET_TX SoftIRQ, the Tx Ring is cleaned of all the descriptors of transmitted packets, that will be de-allocated, and refilled by the packets coming from the Qdisc buffer.

B. The IP lookup mechanism

The IP lookup mechanism included in the Linux kernel is quite complex and complete. In particular, since the 2.1 version, Linux has supported an advanced routing mechanism, which is substantially based on three elements: multiple Routing Tables (RTs), a Routing Policy Database (RPDB) and a Routing Cache, where the most recently used RT entries are stored. While the cache is essential to speed up the look up performance, the presence of multiple RTs and of a RPDB increases the flexibility level of the Linux forwarding functionalities. A flow diagram of the lookup mechanism is reported in Fig. 1. By analyzing in detail the whole mechanism structure, we have to outline that, when determining the route by which to send a packet, the kernel always consults the routing cache first. The routing cache is substantially a hash table used for quick access to recently used routes. If the right entry is found in the routing cache, it will be used to forward the current packet, otherwise, to determine the right route, the kernel will invoke the RPDB. The RPDB is the core element of IP lookup, and can be defined as a cohesive set of route tables and rules. Thus, the main functionalities provided by the RPDB are the mechanism for implementing the rule element of Policy Routing, and the capability to address multiple routing tables. In particular, the RPDB sustains a maximum of 255 routing tables and 2^{32} rules, which potentially corresponds to a rule for each IPv4 address. In such context, a rule may be considered as the filter or selection agent for applying Policy Routing and for choosing the suitable RT for each incoming packet. Rules point to routing tables, several rules may refer to one routing table, and some routing tables may have no rules pointing to them. The RPDB can act on the basis of different parameters, like, for example, the packet source address, the ToS flags, the so called "fwmark" (e.g., the packet filtering tag used by netfilter [17] for matching IP protocols and transport ports), and the reception interface name. For each matching rule in the RPDB, the kernel tries to find a matching route to the destination IP address in the selected RT using a longest prefix match selection algorithm. At this purpose, since kernel 2.6.13, Linux supports two search algorithms: the Radix search tree [18], and the LC-trie [19]. When a matching destination is found, the kernel will select the matching route, and will forward the

packet. If no matching entry is found in the selected routing table, the kernel will pass to the next rule in the RPDB, until it finds a match or falls through the end of the RPDB and all routing tables are consulted.

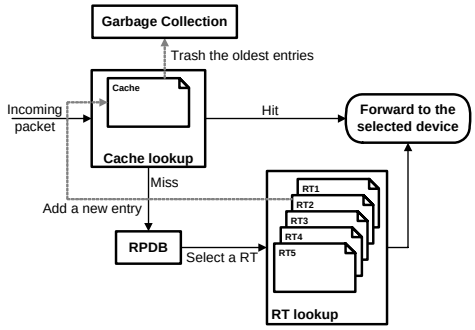

Figure 1. The IP lookup mechanism included in the Linux kernel.

III. System Optimization and Tuning

The whole networking kernel architecture is quite complex and it has many aspects and parameters that should be refined or tuned for a system optimization. For what concerns the parameter tuning, we have used the same configuration adopted in [11]. The 2.6.16.13 kernel images, used to obtain the numerical results in Section V, include four structural patches that we have produced with the aim of testing and/or optimizing the kernel functionalities. In particular, the used patches are described in the following.

A. *Skbuff Recycling patch*

In [20], we studied and realized a new version of the *skbuff* Recycling patch, originally proposed by R. Olsson [21] for the "e1000" driver. In particular, the new version is stabilized for 2.6.16.13 kernel version, and extended to the "sundance" driver. This patch allows to intercept the skbuff descriptors of sent packets before their deallocation, and to re-use them for new incoming packets. As shown in [11], this architectural change heavily reduces the computation weight of the memory management operations, and it allows to achieve a very high performance level (i.e., about the 150-175% of the maximum throughput of standard kernels).

B. *Destination based RT Cache patch*

As previously sketched, the Routing Cache is composed by a hash table, where the most recently used routes are stored. The hash key, computed for each incoming packet, does not univocally match a route, since it is a 32 bit variable that is computed starting from the values of both IP source and destination addresses and of ToS flags. Thus, once the hash key is found, it potentially points to a route list, which is linearly scrolled until all the selectors (i.e., source and destination IP addresses, ToS, fwmarks, incoming device, etc.) agree. When the Policy Routing is not active and routing decisions are taken only on the basis of IP destination values, this kind of cache structure leads to heavy performance limits. For example, as shown in the Section V, when the number of source IP addresses of incoming packets increases, the memory needed by the cache will consistently grows, since a new cache entry is needed at least for each source IP address value. Starting from these considerations, we have decided to propose a new cache architecture to be used without Policy Routing. In this patch the hash key is obtained starting only by the destination address, and only the "fwmark" is used as route selector.

C. *RT Cache disabled patch*

To better evaluate the performance level of the two RT lookup algorithms included in the Linux kernel (i.e., the Radix tree and the LC-trie), we have used a path to disable the RT cache. In this way, all the routes of incoming packets are always computed by these algorithms.

D. *Performance Counter patch*

To study in depth the OR internal behaviour, we have decided to introduce a set of counters in the kernel source code, with the aim of understanding how many times a certain procedure is called, or how many packets are kept per time. In detail, we have introduced the following counters: *IRQ*: number of interrupt handlers generated by a network card; *tx/rx IRQ*: number of tx/rx IRQ routine per device; *tx/rx SoftIRQ*: number of tx/rx software IRQ routines; *Qdiscrun* and *Qdiscpkt*: number of times when the output buffer (Qdisc) is served, and number of served packets per time; *Pollrun* and *Pollpkt*: number of times when the rx ring of a device is served, and the number of served packets per time; *tx/rx clean*: number of times when the tx/rx procedures of the driver are activated.

IV. Benchmarking Tools

To benchmark the OR forwarding performance, we have used a professional equipment, namely Agilent N2X Router Tester [22], which allows to obtain throughput and latency measurements with very high availability and accuracy levels (i.e., the minimum guaranteed timestamp resolution is 10 ns).

To better support the performance analysis and to identify the OR bottlenecks, we have also performed some internal measurements by using specific SW tools (called profilers) placed inside the OR, which are able to trace the percentage of CPU utilization for each SW modules running on the node. The problem is that many of these profilers require a relevant computational effort that perturbs the system performance. We have experimentally verified that one of the best is Oprofile [23], an open source tool that realizes a continuous monitoring of system dynamics with a frequent and quite regular sampling of CPU HW registers. Oprofile allows the effective evaluation of the CPU utilization of both each SW application and each single kernel function with a very low computational overhead.

V. Numerical Results

In this Section, we report some of the numerical results carried out to evaluate the performance of IP lookup mechanism. We propose two different benchmarking test sessions: one for evaluating RT cache scalability and performance, the other to characterize the performance and to take a comparison between the Radix and the LC-trie lookup mechanisms. For each benchmarking session, we show three different kinds of results: external measurements (i.e., throughput and average latency), profiling statistics and performance counters (i.e., the kernel patch in Section III.D). While the external measurements are the classical performance indexes used to benchmark network devices, both the profiling and the internal counters allow us to evaluate OR internal dynamics and features, and to characterize its SW architecture bottlenecks. About the profiling results, we have classified kernel functions in different logical sets, which correspond to the main high level kernel functionalities, like idle, kernel scheduler, memory management, IP processing, NAPI, TxAPI,

HW IRQ routines, Ethernet Processing and oprofile functions. Moreover, this classification has been lightly changed with respect to other our works on the OR.

In all the benchmarking sessions here reported, we have used a traffic matrix composed by a single flow with 64 sized datagrams that crosses the OR between two different Gigabit Ethernet adapters. Moreover, we have chosen to use small sized datagrams, since the computational capacity is the actual performance bottleneck in the OR, as shown in [11]. About the HW architecture, we have used a system based on a Supermicro X5DL8-GG mainboard, 2.4 Ghz Intel Xeon processors, and Intel PRO 1000 XT Server network adapters.

A. Benchmarking tests on RT Cache

The RT cache is characterized by a per flow structure, since the 32 bit hash key is built according the source and destination IP addresses and the ToS value.

While the per flow cache structure is clearly needed if we use advanced routing policies, when routing decisions are made only on the basis of destination IP address, it reduces the scalability level. At this purpose, Figs. 2 and 3 show the throughput and the latency values obtained with a optimized kernel (including the *skbuff* recycling patch) in presence of a traffic flow composed by datagrams with different source IP addresses.

Observing these Figs. we can note how the maximum throughput value decreases according the raising of source address number, and according the linear increase of cache dimension, as shown in Fig. 4. Moreover, as we can observe by Fig. 3, when the OR reaches the saturation (over the 50% of offered load), the average delay increases according the raise of source IPs: this is clearly due to the increase of searching times in the RT cache.

Figs. 5 and 6 show the profiling results for the 1 and 16384 source IP addresses tests (with reference to Fig.2): in this case, we can outline how, coherently with expectations, the IP processing functions double their whole computational weight. The behaviour of the other functionalities is clearly bound with the number of forwarded packets: the weight of almost all the classes raises linearly up to the saturation point, and after it remains more or less constant. On the contrary IRQ handlers show a particular behaviour, caused by the NAPI paradigm: when the traffic load is low their computational weight is high, since the Rx API works like an interrupt mechanism; while, for higher loads, it starts to lower more and more. When IRQ weight becomes zero, the OR reaches the saturation point, and works like polling mechanism. This is confirmed also by the performance counters reported in Figs. 7 and 8, where we can view both the tx and rx boards reducing their IRQ generation rates, while the kernel passes from polling the rx ring twice per received packet, to about 0.22 times. Also the number of Rx SoftIRQ per received packet decreases as offered traffic load raises.

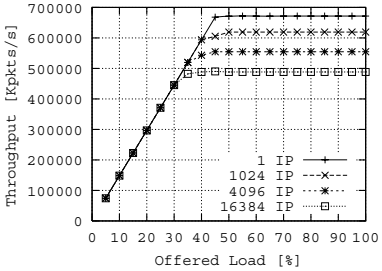

Figure 2. Throughput values of the kernel with skbuff recycling patch and a variable number of IP source addresses.

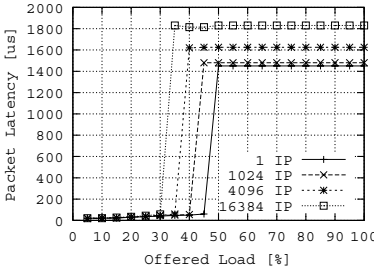

Figure 3. Average latency values of the kernel with skbuff recycling patch and a variable number of IP source addresses.

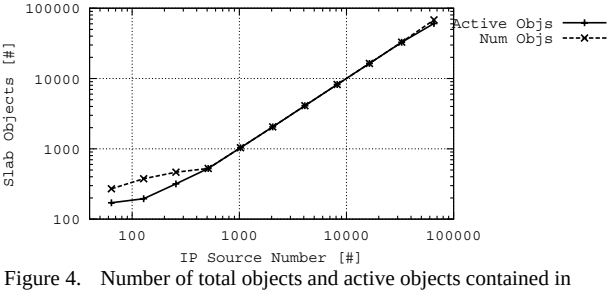

Figure 4. Number of total objects and active objects contained in the RT cache for the kernel with skbuff recycling patch, the offered load is 1Gbps.

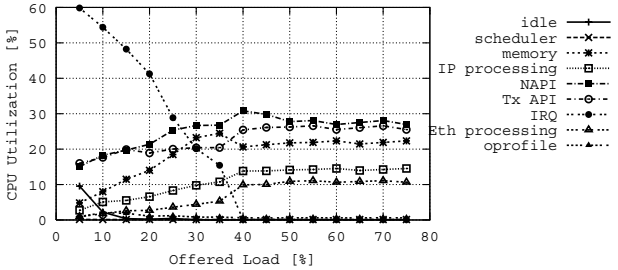

Figure 5. Profiling results of the skbuff recycling patched kernel, in presence of an incoming traffic flow with a single IP source address.

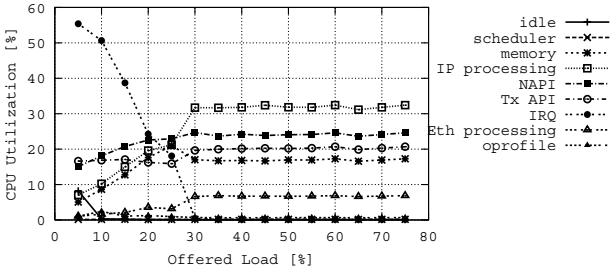

Figure 6. Profiling results of the skbuff recycling patched kernel, in presence of an incoming traffic flow with 16384 IP source address values.

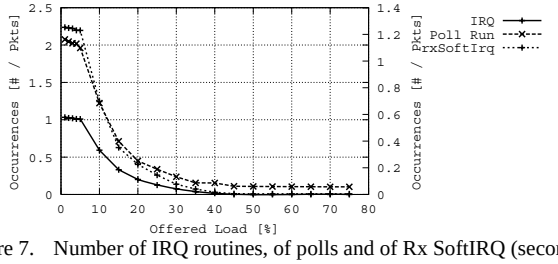

Figure 7. Number of IRQ routines, of polls and of Rx SoftIRQ (second y-axe) for the RX board for the skbuff recycling patched kernel, in presence of an incoming traffic flow with only 1 IP source address.

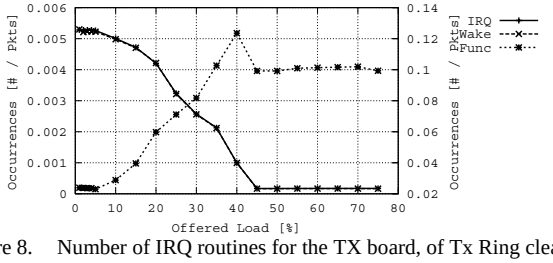

Figure 8. Number of IRQ routines for the TX board, of Tx Ring clean by TxSoftIRQ (“func”) and by RxSoftIRQ (“wake”) (second y axe)for the skbuff recycling patched kernel, in the same conditions of Fig. 7.

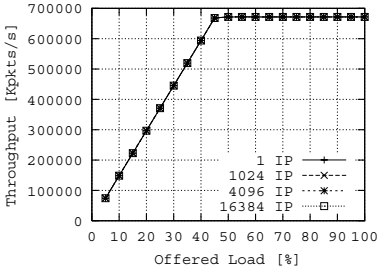

Figure 9. Throughput values of the kernel with skbuff recycling and cache patch, according a variable number of IP source addresses.

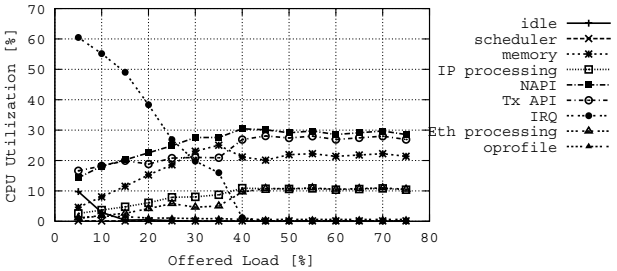

Figure 10. Profiling results of the skbuff recycling and cache patched kernel, in presence of an incoming flow with 16384 IP source addresses.

For what concerns the transmission dynamics, Fig. 8 shows very low function occurrences: with reference to Fig. 8, the Tx IRQ routines low their occurrences up to the saturation, while the "wake" function, which represents the number of times that the Tx Ring is cleaned and the Qdisc buffer is served during a Rx SoftIRQ, shows a mirror behaviour: this because when the OR reaches the saturation, all the tx functionalities are activated when the Rx SoftIRQ starts. Fig. 9 reports the throughput values obtained with the cache patch described in Section III.B: as you can observe, with a destination based cache and using no advanced routing policies, there is not any performance decays according different source IP address ranges. In fact, in all the performed tests, the number of RT cache entries is resulted to be low and constant, guaranteeing, in this way, very fast cache lookup times (all the obtained results look very similar to the standard case with only 1 source IP address).

B. IP lookup benchmarking

In this second test session, we show a comparison between the two IP lookup mechanisms included in the Linux kernel: the Radix tree and the LC-trie. At this purpose, in all the tests reported in this Section, we use a traffic flow composed by a variable range of random IP destination addresses that hit uniformly the RT. Thus, we have performed a first test, using a skbuff recycling patched kernel, with the aim of analyzing the OR scalability level, when the number of IP destination addresses rises. Like in the case of Fig. 2, an arising number of IP addresses causes an enlargement of the RT cache, and a consequent increase in searching times. This is clearly observable in Fig. 12, where the obtained throughput values show a very similar behaviour to those in Fig. 2: the performance decay is not caused by the real IP lookup mechanism (since it activated only for the first datagram with a new IP destination address, and afterwards a new cache entry will be allocated and used), but it is substantially due to the searching time increase in the cache table hash, in presence of 256, 1024, 8192 and 16384 entries, respectively. Therefore, to better analyze and compare the performance of the Radix and LC-trie IP lookup mechanisms, we have decided to disable the cache with the kernel patch introduced in Section III.C. In such environment, we have repeated the previous test for both the mechanisms, and Figs. 11, 12 , 13 and 14 show the external measurement results. The Radix mechanism appears to have a lower computational complexity with respect to the LC-trie, since it allows higher throughputs (i.e., the difference is more or less 30-50 kpps) and lower packet latencies. The profiling results, shown in Figs. 15, 16, 17 and 18, confirm this trend, as IP functions have lower CPU utilization for the Radix kernel. However this performance difference seems to reduce as the RT entry number rises.

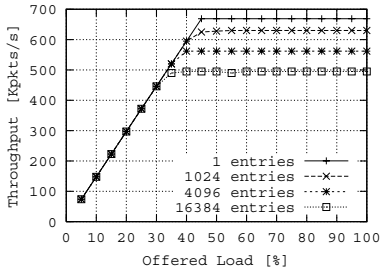

Figure 11. Throughput values of the kernel with skbuff recycling patch, Radix lookup and with variable sized RT.

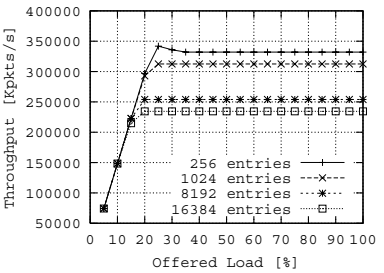

Figure 12. Throughput values of the kernel with skb recycling and cache patches, Radix lookup and with RT with variable size.

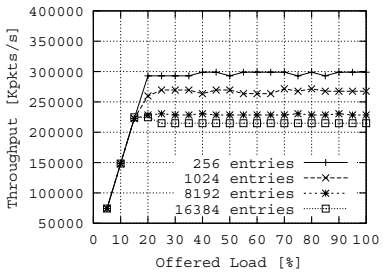

Figure 13. Throughput values of the kernel with skbuff recycling and cache disabling patches, LC-trie lookup and with RT with variable size.

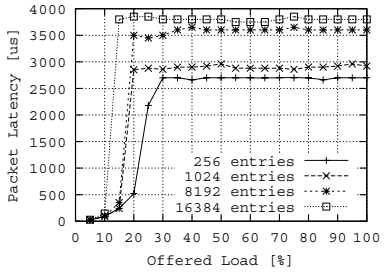

Figure 14. Latency values of the kernel with recycling and cache patches, LC-trie lookup and with RT with variable size.

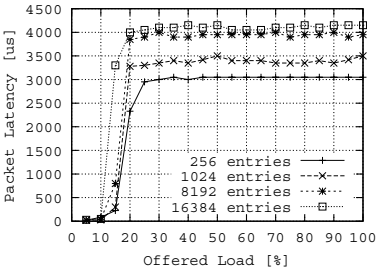

Figure 15. Latency values of the kernel with recycling and cache patches, Radix lookup and with RT with variable size.

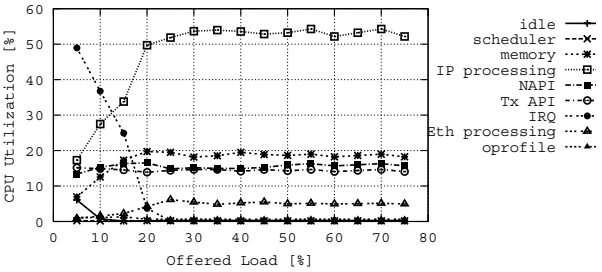

Figure 16. Profiling results of the skbuff recycling and disabling cache patched kernel with Radix lookup, in presence of 256 RT entries.

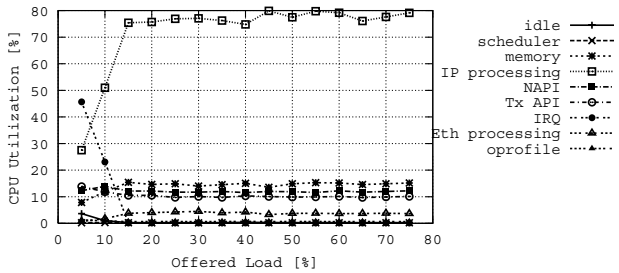

Figure 17. CPU Utilization of the most significant kernel functionalities involved in the forwarding process for the skbuff recycling and disabling cache patched kernel with Radix lookup, in presence of 16384 RT entries.

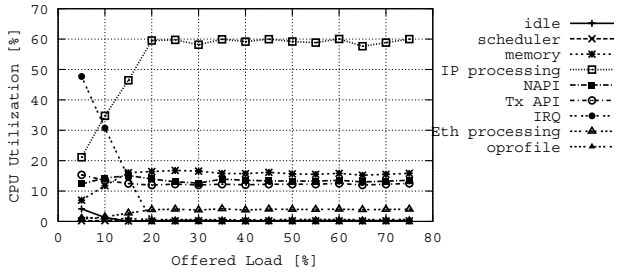

Figure 18. CPU Utilization of the most significant kernel functionalities involved in the forwarding process for the skbuff recycling and disabling cache patched kernel with LC-trie lookup, in presence of 256 RT entries.

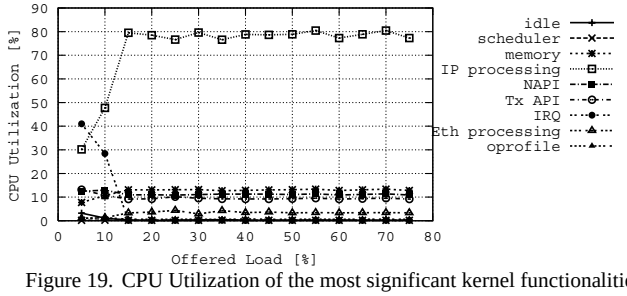

Figure 19. CPU Utilization of the most significant kernel functionalities involved in the forwarding process for the skbuff recycling and disabling cache patched kernel with LC-trie lookup, in presence of 16384 RT entries.

VI. Conclusions

In this work, we have introduced a complete study of the IP lookup mechanism included in the Linux kernel: we have shown a detailed performance evaluation, using both internal and external measurements, and some kernel patches to optimize its behavior. In particular, this work offers two main contributions: the scalability level of the per flow RT cache has been tested and compared with a destination based structure, and the Radix and the performance of LC-trie lookup algorithms have been compared and discussed.

References

[1] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek,"*The Click modular router*", ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297.

[2] Zebra, http://www.zebra.org/.

[3] M. Handley, O. Hodson, E. Kohler, "*XORP: an open platform for network research*", ACM SIGCOMM Computer Communication Review, Vol. 33 Issue 1, Jan 2003, pp. 53-57.

[4] A. Bianco, R. Birke, D. Bolognesi, J. M. Finochietto, G. Galante, M. Mellia, M.L.N.P.P. Prashant, Fabio Neri, "*Click vs. Linux: Two Efficient Open-Source IP Network Stacks for Software Routers*", Proc. of IEEE 2005 Workshop on High Performance Switching and Routing (HPSR 2005), Hong Kong, May 12-14, 2005, pp. 18-23.

[5] A. Bianco, J. M. Finochietto, G. Galante, M. Mellia, F. Neri, "*Open-Source PC-Based Software Routers: a Viable Approach to High-Performance Packet Switching*", Third International Workshop on QoS in Multiservice IP Networks, Catania, Italy, Feb 2005, pp. 353-366.

[6] A. Bianco, R. Birke, G. Botto, M. Chiaberge, J. Finochietto, M. Mellia, F. Neri, M. Petracca, G. Galante, "*Boosting the Performance of PC-Based Software Routers with FPGA-enhanced Network Interface Cards*", Proc. of IEEE 2006 Workshop on High Performance Switching and Routing (HPSR 2006), Poznan, Poland, June 2006, pp. 121-126.

[7] B. Chen and R. Morris, "*Flexible Control of Parallelism in a Multiprocessor PC Router*", Proceedings of the 2001 USENIX Annual Technical Conference (USENIX '01), Boston, Massachusetts, June 2001.

[8] C. Duret, F. Rischette, J. Lattmann, V. Laspreses, P. Van Heuven, S. Van den Berghe, P. Demeester, "*High Router Flexibility and Performance by Combining Dedicated Lookup Hardware (IFT), off the Shelf Switches and Linux*", Proc. of Second International IFIP-TC6 Networking Conference, Pisa, Italy, May 2002, LNCS 2345, Ed E. Gregori *et al*, Springer-Verlag 2002, pp. 1117-1122.

[9] A. Barczyk, A. Carbone, J.P. Dufey, D. Galli, B. Jost, U. Marconi, N. Neufeld, G. Peco, V. Vagnoni, "*Reliability of datagram transmission on Gigabit Ethernet at full link load*", LHCb technical note, LHCB 2004-030 DAQ, March 2004.

[10] Building Open Router Architectures based on Router Aggregation (BORA BORA) project, www.telematica.polito.it/projects/borabora/.

[11] R. Bolla, R. Bruschi, "*RFC 2544 Performance Evaluation and Internal Measurements for a Linux Based Open Router*", Proc. of IEEE 2006 Workshop on High Performance Switching and Routing (HPSR 2006), Poznan, Poland, June 2006, pp..9-14.

[12] R. Bolla, R. Bruschi, "*A high-end Linux based Open Router for IP QoS networks: tuning and performance analysis with internal (profiling) and external measurement tools of the packet forwarding capabilities*", Proc. of the 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurements (IPS MoMe 2005), Warsaw, Poland, Mar. 2005, pp. 203-214.

[13] R. Bolla, R. Bruschi, "*IP forwarding Performance Analysis in presence of Control Plane Functionalities in a PC-based Open Router*", Proc. of the 2005 Tyrrhenian International Workshop on Digital Communications (TIWDC 2005), Sorrento, Italy, Jun. 2005, and in F. Davoli, S. Palazzo, S. Zappatore, Eds., "Distributed Cooperative Laboratories: Networking, Instrumentation, and Measuments", Springer, Norwell, MA, 2006, pp. 143-158.

[14] J. Fu, O. Hagsand, G. Karlsson, "*Performance Evaluation and Cache Behavior of LC-Trie for IP-Address Lookup*", Proc. of IEEE 2006 Workshop on High Performance Switching and Routing (HPSR 2006), Poznan, Poland, June 2006, pp. 29-36.

[15] K. Wehrle, F. Pählke, H. Ritter, D. Müller, M. Bechler, "The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel", Pearson Prentice Hall, Upper Saddle River, NJ , USA, 2004.

[16] J. H. Salim, R. Olsson, A. Kuznetsov, "*Beyond Softnet*", Proc. of the 5th annual Linux Showcase & Conference, November 2001, Oakland California, USA.

[17] The NetFilter project, homepage at www.netfilter.org.

[18] S. Nilsson, G. Karlsson, "*IP-address lookup using LC-tries*", IEEE Journal on Selected Areas in Communications, vol. 17, no. 6, pp. 1083-1092, Jun. 1999.

[19] D. R. Morrison, "*PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric*", Journal of the ACM (JACM), vol. 15 , no. 4, pp. 514 – 534, Oct. 1968.

[20] Open Router resources from the TNT Lab., homepage at www.tnt.dist.unige.it.

[21] The descriptor recycling patch, ftp://robur.slu.se/pub/Linux/net-development/skb_recycling/.

[22] The Agilent N2X Router Tester, http://advanced.comms.agilent.com /n2x/products/index.htm

[23] Oprofile, http://oprofile.sourceforge.net/news/.