Towards Low Latency Software Routers

Torsten M. Runge[a], Daniel Raumer[b], Florian Wohlfart[b], Bernd E. Wolfinger[a], Georg Carle[b]
[a] Universität Hamburg, Dept. of Computer Science, Telecommunications and Computer Networks
Email: {runge, wolfinger}@informatik.uni-hamburg.de
[b] Technische Universität München, Dept. of Computer Science, Network Architectures and Services
Email: {raumer, wohlfart, carle}@net.in.tum.de

Abstract— **Network devices based on commodity hardware are capable of high-speed packet processing while maintaining the programmability and extensibility of software. Thus, software-based network devices, like software routers, software-based firewalls, or monitoring systems, constitute a cost-efficient and flexible alternative to expensive, special purpose hardware. The overall packet processing performance in resource-constrained nodes can be strongly increased through parallel processing based on off-the-shelf multi-core processors. However, synchronization and coordination of parallel processing may counteract the corresponding network node performance. We describe how multi-core software routers can be optimized for real-time traffic by utilizing the technologies available in commodity hardware. Furthermore, we propose a low latency extension for the Linux NAPI. For the analysis, we use our approach for modeling resource contention in resource-constrained nodes which is also implemented as a *resource-management* extension module for ns-3. Based on that, we derive a QoS-aware software router model which we use to evaluate our performance optimizations. Our case study shows that the different scheduling strategies of a software router have significant influence on the performance of handling real-time traffic.**

Index Terms— **delay, latency, parallel processing, resource contention, scheduling strategy, software router, simulation**

I. Introduction

PROGRAMMABILITY and extensibility of commodity hardware has attracted use cases [1] where software has to be able to process network traffic with performance equivalent to hardware routers [2]. Rapid deployment of new features in the programmable software-based data plane is a driver of the software-defined networking paradigm [3]. Previous works have shown different bottlenecks that limit the achievable throughput of software-based packet processing systems in terms of bits or packets per second [2], [4], [5]. Some of these bottlenecks can be mitigated by efficient networking software [6]–[8]. Others require differentiated distribution of operations to hardware resources [9]. However, new applications of commodity hardware in routers, monitoring systems, or next generation firewalls introduce new challenges for packet treatment to software development. These systems have to differentiate packets and treat packet flows differently if they are critical in terms of Quality of Service (QoS) parameters like throughput, delay, jitter, packet loss, or connection establishment time [10].

This paper is an extended version of our previously presented work [11]. We investigate different scheduling strategies for low latency packet processing in software routers by making use of the underlying hardware. Therefore, we analyze the usage of dedicated Rx rings to reduce packet latencies of specific traffic by applying our low latency support for software routers. In Section II, we discuss the related work. Section III explains packet processing steps in a software router and shows approaches for performance improvements. In Section IV we propose an efficient and resilient architecture for a QoS-aware software router. In Section V, we create a detailed model of the proposed architecture that allows us to evaluate our low latency concept based on simulation results. The software router model is used in a case study in Section VI to show how software router performance can be improved. We summarize the paper and highlight our contributions in Section VII.

II. Related Work

With Netmap [6], PF_Ring [7], and Intel DPDK [8] three techniques exist that optimize the software side of PC-based packet processing. These frameworks significantly increase the packet processing performance through avoiding unnecessary context switches between the kernel and the user space by melting driver, kernel, and even application parts of the packet processing chain.

Besides, hardware features like Direct Cache Access (DCA) [12], which has already developed into a standard technique in servers, further help to improve the performance. Other examples of advanced hardware features like the *Intel Flow Director* feature, that filters and classifies packets in hardware based on almost arbitrary match fields [13], can be found in current network interface cards (NIC) like the Intel X540 [14]. By making use of these techniques, Tanyingyong et al. presented a case study and implemented a fast routing path, where the routing decision for a limited number of flows (typically those with high packet rates) is offloaded to the NIC [15].

Know-how on measurement practices was already described in 2005 by Tedesco et al. who published a

Manuscript received November 28, 2014; revised February 20, 2015; accepted April 1, 2015. © 2015 IEEE.

This work was supported by the German Research Foundation (DFG) as part of the *MEMPHIS* project and the German Federal Ministry of Education and Research (BMBF) under EUREKA project SASER.

technique to measure different parts of PC-based packet processing systems with commodity hardware [16] based on a simple understanding of software router internal queueing: They measured mean delays of $5\,\mu$s, $20\,\mu$s, and $5\,\mu$s for input queueing, processing, and output queueing using a 2.4 GHz single-core CPU at maximum lossfree load. Carlsson et al. presented a delay measurement setup for routers as black boxes [17] that follows RFC 2679 [18], which specifies guidelines to measure one-way packet delay. They measured a mean delay of $98\,\mu$s for UDP traffic and $101\,\mu$s for ICMP traffic for the tested hardware routers. Compared to the $30\,\mu$s obtained by Tedesco et al. [16], these delays are significantly higher. In particular, they observed a long tail distribution of packet delays. As software-based packet generators are not accurate enough to deliver reliable measurements of delays [19], thus packet generators with hardware support (e.g. [20]) are necessary for accurate measurements [21]. Rotsos et al. [22] utilized FPGAs for accurate software switch delay measurements. In 2007, Bolla and Bruschi presented a detailed study of a single core software router based on Linux kernel 2.6 and performed RFC 2544 conform tests with a special network device testing box [23]. The dedicated device testing box allowed to measure delays with microsecond accuracy. Depending on the type of software router, the configuration, and the packet size they measured delays from $14\,\mu$s to hundreds of μs. In scenarios where the CPU was the bottleneck, the delay increased to more than 16 ms. In 2008, Bolla and Bruschi presented a study of architectural bottlenecks in software and hardware. They described and evaluated different uses of multiple Rx and Tx rings as these have been available [4]. A newer study of software router performance [2] and a study of performance based on different router workloads were published by Dobrescu et al. [5]. Recently, we studied the performance of software routers in a multi-core setup [24].

Analytical modeling and simulation are also widely-used in the networking research area as a cost-effective approach to design, validate, and analyze protocols and algorithms in a controlled and reproducible manner. Begin et al. [25] developed a high-level approach to model an observed system behavior with little knowledge about the system internal structure or operation by adequately selecting the parameters of a set of queueing systems and queueing networks. Dobrescu et al. [5] described an analytical cache model for cache misses with multiple, well defined parallel packet flows on a multi-core software router assuming infinite queue sizes. However, in the research community queueing systems with finite capacity are seen as being more precise than those with infinite queues [26]. Chertov et al. presented a device-independent router model that takes into account the queue size and the number of CPU cores inside a router [27]. With specific parameters, this model can be used for different router types (e.g. Cisco 7602).

Nonetheless, discrete event simulations outperform analytical models with respect to more detailed modeling and realistic traffic load scenarios which include bursty traffic, different packet attributes (e.g packet size), variable packet inter-arrival, and service times. There exist a variety of commercial and open source network simulation tools [28], [29] such as ns-2 [30], ns-3 [31], OMNeT++ [32] and OPNET [33]. Moreover, there are also open source system simulators such as gem5 [34], MARSS [35], and Sniper [36]. These simulators provide accurate results based on fine-grained node models which are mainly used for the development of microprocessor architectures. However, the high level of detail compromises scalability and imposes a large modeling effort. Especially for a network researcher, this makes it difficult to get a high-level understanding of the system behavior with respect to packet processing.

With *nsclick* [37] the router software *Click Modular Router* [38] was combined with the network simulator ns-3 [31] which enables the easy transfer of real code from the testbed to the simulation model. Wu et al. [39] presented a task allocation concept for packet processing nodes where a task is represented as one-to-one mapping of a Click element. Thus, they optimized the configuration of a multi-core Click router during its runtime by duplication of dedicated bottleneck tasks. Kristiansen et al. [40] proposed a node model for considering the packet processing overhead resulting from software. However, this model does not consider multi-core architectures. Previously, we presented a modeling approach for a detailed node model [41] of a packet processing system based on multi-core CPUs and other system internal components (e.g. memory, network, I/O) that we will use to conduct the case study in this paper.

III. Software Router Architecture

The Internet architecture was designed based on the best effort approach. This works well for traditional applications of the Internet (e.g. e-mail, file transfer, world wide web) which are less time-critical. However, today the Internet is also increasingly used for real-time communications such as IP telephony (VoIP), video conferencing and online gaming. Real-time applications are very sensitive to packet latencies but can often handle a certain amount of packet loss without degrading the user experience due to failure correction mechanisms on higher layer. Thus, they have specific QoS constrains.

For instance, if the one-way latency in a VoIP telephone conference becomes greater than 150 ms the user experience is perceived as unacceptable [10], [42] whereas the same latency for a file transfer is unproblematic.

Real-time traffic constitutes a challenge for the existing Internet infrastructure. In particular, Internet routers need to distinguish between traffic classes (e.g. real-time and best-effort) to prefer the real-time traffic with low latency constraints. Especially, real-time packet processing becomes even more challenging for software routers which are described in the following.

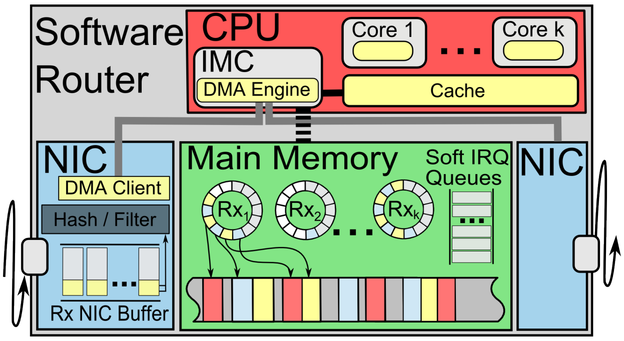


Figure 1. Hardware resources in a software router

A. *Packet Processing in Software Routers*

a) Commodity Hardware Processing: Software routers are based on commodity hardware with general purpose CPUs where the packet processing is implemented in software routines. In contrast, hardware routers accelerate performance-critical packet processing functions (data plane processing) by using special purpose chips. For more complex processing tasks (usually control plane functions, e.g. routing protocols) even hardware routers rely on a general purpose CPU. Implementing complex processing in software is both easier to realize and has a much shorter development cycle. Usually control plane functions are not required to process packets in line rate and are often synthetically limited to avoid uncontrolled overloading of the CPU. For instance, this ensures that a router can still process an ICMP ping even if it would be overloaded with SNMP requests.

Several improvements have been implemented for off-the-shelf hardware to cope with high-speed networks by efficiently exploiting multi-core CPUs. Fig. 1 shows resources that are relevant for software routers when processing packet flows. An integrated memory controller (IMC) was included in the CPU which provides a Direct Memory Access (DMA) engine to the PCIe connected components. Even preemptive copying of data into the caches (Direct Cache Access; DCA) [12] is common today.

b) NIC Hardware Processing: Modern NICs provide features like receive side scaling (RSS) and segmentation offloading to shift packet processing tasks from the CPU to the NIC controller and to distribute the tasks efficiently among the CPU cores. Packets arriving at the ingoing interface are stored in the *Rx NIC Buffer*. NIC controllers like the Intel 82599 or X540 support different programmable hardware filters and hash-based RSS [43], [14] which are applied in a hierarchical order.

Hardware filters of the NIC allow to match header fields like IP/MAC addresses, ports, VLAN tags, or arbitrary protocols and flags. On multi-queue NICs these hardware filters can be used to enqueue incoming packets to a certain queue for received packets (Rx ring) based on specific packet attributes (aka. traffic classification). The last layer of the filters is RSS, which uses a static hash on header fields to assign packets to Rx rings and thus to the CPU cores. With these filters NICs are able to efficiently distribute the incoming packet processing workload across multiple CPU cores [13]. This also ensures that each packet of a specific flow is served by the same CPU core which avoids packet reordering and context switches between CPU cores. After traffic classification, the packets get transferred on behalf of the DMA engine via the PCIe bus to the main memory. A DMA client can access a DMA provider (software) and the related DMA engine (hardware) to write and read data from DMA channels. This allows the NIC as DMA client to copy data to the main memory without involvement of the CPU.

c) NIC Driver and Linux Kernel Processing: Since Linux kernel 2.5 the New API (NAPI) defines an interrupt moderation mechanism for the packet reception and transmission [23], [44], [45]. On packet reception, the DMA engine triggers a hardware interrupt (IRQ) for the CPU core assigned to the Rx ring after the packet was copied to the Rx ring in the main memory. The Rx rings store descriptors, which contain pointers to the actual packets that reside in an unordered manner in the main memory. An IRQ can be assigned to a specific CPU core which may handle one or multiple Rx rings. If multiple Rx rings are assigned to a CPU core then the Rx rings are handled in a round robin manner. The IRQ schedules a so-called "soft IRQ" `net_rx_softirq` in the OS which handles all time-consuming tasks later. If the corresponding soft IRQ of the Rx ring is already scheduled to process incoming packets, then IRQs of this Rx ring are disabled. When the soft IRQ `net_rx_softirq` gets executed by the OS scheduler, the CPU core fetches packets from the Rx ring (aka. polling) to process them.

Depending on the application (user or kernel space) further copy overhead can be necessary for each context switch. For instance, in order to forward a packet, the router needs to perform a lookup in the forwarding table, update the TTL (or hop count) field in the IP header, and trigger the sending process on the outgoing interface. If the packet is addressed to the router itself, such as in case of routing protocol updates handled by XORP [46] or Quagga [47], the packet processing is more complex, however not as time-critical as in the case of packet forwarding. After processing a certain number of packets (aka. budget), the polling is suspended (and the corresponding soft IRQ must be rescheduled) to avoid permanent blocking of the CPU core. The polling terminates when all packets of this Rx ring have been processed. Then, the CPU core is released and the IRQ for packet reception is reenabled.

Finally, for packet transmission, the descriptor is placed in the corresponding queue for outgoing packets (Tx ring) of the outgoing interface. After the NIC successfully transmits one or multiple packets, it generates a IRQ which schedules a soft IRQ `net_tx_softirq` to transmit the new packets.

For further details, we investigate the packet latency caused by software routers based on testbed measurements and simulations [48].

B. QoS in the Linux Kernel

In addition to the ring size for limiting the number of packets in the ring, Byte Queue Limits (BQL) were introduced with the Linux kernel 3.3.0 which defines a limit on the number of Bytes in the Tx ring buffer. Without BQL a Tx ring buffer contains an unknown amount of payload data depending on the Tx ring size and the actual packet sizes. Besides, the packet size can also be a multiple of the maximum transmission unit (MTU) due generic segmentation offload (GSO). The GSO mechanism enables the NIC to split and merge packets. Thus, there can be packets in the Tx ring which are bigger than the MTU. By limiting the number of Bytes in the Tx rings the application of differentiated packet treatment is delayed to the different queueing disciplines (qdisc). In contrast to the BQL, the qdiscs are more flexible as they are implemented in the kernel only and do not require any support by the driver.

TABLE I.
QDISC STRATEGIES IN LINUX

	Classful	Reordering	Shaping
pfifo_fast	No	No	No
Token Bucket Filter (TBF)	No	No	No
Stochastic Fair Queueing (SFQ)	No	fair	No
Extended SFQ (ESFQ)	No	fair	No
Random Early Detection (RED)	No	No	dynamic
Hierarch. Token Bucket (HTB)	Yes	implicit	implicit
Hierarch. Fair Service Curve (HFSC)	Yes	fair	implicit
Priority scheduler (PRIO)	Yes	explicit	implicit
Class Based Queueing (CBQ)	Yes	implicit	implicit

Qdiscs are techniques for differentiated packet treatment in the Linux kernel. Filtering can be applied to ingress traffic of the Linux kernel and even more complex mechanisms which also include reordering to the egress or parts of the egress traffic of the Linux kernel. Table I shows existing queueing disciplines in Linux. All classful qdiscs can be applied to selected classes of traffic. Depending on the applied qdisc some packets are transmitted earlier than if they would be transmitted with the standard first-come-first-served (FCFS) queueing behavior. Which techniques may change the order of packets when applied can be seen in Table I. Some algorithms cause packet reordering due to the goal of a fair bandwidth distribution to more competitors. The HTB and the CBQ qdisc implicitly reorder packets depending on the configuration as these are classful. PRIO explicitly reorders packets due to different prioritization. However, to have the best effect on the QoS of traffic passing a component each of these mechanisms must be applied before the bottleneck. Thus, with the described techniques, traffic shaping on a software router can avoid congestion of the outgoing Internet connection but cannot avoid service degradation due to an overloaded software router CPU. Consequently this is also valid for techniques like DiffServ [49] or IntServ [50] that cannot deal with an overloaded CPU core.

IV. QoS-AWARE SOFTWARE ROUTER ARCHITECTURE

Previous work described how software routers have to be configured to utilize numerous Rx and Tx rings for efficient load balancing to different cores, but did not consider QoS differentiation [4]. Implementations of state-of-the-art QoS differentiation techniques in the Linux kernel (and other software routers) are well-suited for home routers and other scenarios where the link capacity is the bottleneck. However, we argue that these implementations do not work well enough in scenarios where the software router is the bottleneck rather than the egress link. In a software router the CPU is the main bottleneck [2], [5], [24], as other components such as the main memory and system buses usually handle significantly higher bandwidths than the CPU can process. In case the incoming traffic is overwhelming the CPU, such that it cannot process all incoming packets, as soon as the Rx ring is filled some of the incoming packets are already dropped before being processed by the CPU. In this case, the approach to add traffic classification as just another step during the general packet processing does not work, because high-priority packets might have already been dropped before this processing step. This means, incoming packets need to be classified before being processed in the CPU, in order to provide QoS differentiation (e.g. upper bound for packet latency).

There are several possible approaches to solve this problem: One approach is to use one or multiple cores (as many cores as necessary to classify any type of incoming traffic at line speed) for receiving, classifying and forwarding the incoming traffic to the other cores for actual packet processing. This approach can be realized using PF_RING DNA clusters [51] (where the library *libzero* implements clusters to distribute incoming packets to multiple applications or threads) or Receive Packet Steering (RPS) [52] (which is a software implementation of RSS).

Another approach is to reduce the required number of CPU cycles per packet treatment by offloading the traffic classification into the NIC with multi-queue support. The NIC allocates dedicated Rx rings for specific traffic classes. Additionally, each Rx ring requires a priority corresponding to its traffic class (e.g. real-time traffic). Finally, the corresponding soft IRQs of the dedicated Rx rings need to be scheduled by the OS with the help of an optimal scheduling strategy.

We favor our latter approach, because it efficiently exploits today's commodity hardware and can be implemented in the Linux NAPI. In Fig. 2 this approach is illustrated which is called *Low Latency* (LL) extension for software routers. The receiving NIC (a) classifies incoming packets into multiple Rx rings based on its packet attributes (b) per core (c), which enqueues the processed packets into a Tx ring (d) of the egress NIC (e). An exclusive Tx ring for each combination of NIC and core allows to omit locking mechanisms.

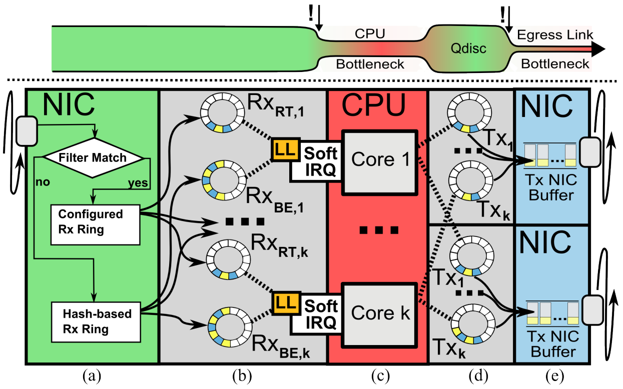


Figure 2. QoS-aware software router with low latency support before the CPU bottleneck

A. Traffic Classification in the NIC

Fig. 2 shows a generic form of classifying traffic at the receiving NIC into multiple Rx rings with different priorities per core. $Rx_{RT,i}$ refers to the high priority real-time (RT) traffic rings and $Rx_{BE,i}$ relates to best effort (BE) rings where i denotes the CPU core to which the Rx ring is pinned. We use two priority classes to demonstrate our approach, however our concept in general is not limited to two priority classes. In the following we refer to the specific features of the Intel X540 Ethernet controller [14] as an example, although the described features are not new to this controller and can also be found in older Ethernet controllers like the Intel 82599 Ethernet controller [43]. This Ethernet controller offers more than one feature that can be used to implement our strategy, however each of them has a different application scenario. If the incoming traffic already comes with priority labels in their IEEE 802.1Q VLAN tags, a combination of the data center bridging (DCB) feature with receive side scaling (RSS) can automatically classify received packets into multiple Rx ring queues per core. If the incoming traffic does not carry priority tags, prioritized packets need to be identified using header information, such as IP addresses or port numbers. The Intel X540 Ethernet controller supports different types of hardware filter rules, which can be used to match packet header fields and explicitly sort matched packets into a specified Rx ring. The packets that do not match any of these filter rules are put into the default best effort Rx rings that are distributed among all cores via the RSS feature. For instance, the Intel X540 Ethernet controller can match the following header fields: VLAN header, IP version (IPv4, IPv6), source and destination IP address, Transport layer protocol (UDP, TCP, SCTP), source and destination port number, or flexible 2-Byte tuples within the first 64 Bytes of the packet (e.g. applicable to match the TOS field). These filters provided by the NIC are able to offload packet matching from the CPU to the NIC and thus increase performance [13].

B. Scheduling Strategy for Prioritized Rx Rings

In the state-of-the-art, multiple Rx rings are handled in a round robin manner which supports no prioritized packet processing. For our proposed prototype the Linux NAPI has to be extended that it supports a scheduling strategy which considers multiple Rx rings with priorities.

Without any significant computational overhead it is possible to implement a priority that only processes packets from an Rx ring if all higher prioritized rings are empty. Therefore, this approach is more flexible than dedicating one or more cores exclusively to prioritized traffic, which results in wasted clock cycles if the prioritized traffic does not fully utilize all dedicated cores. Other scheduling strategies, such as weighted fair queueing (WFQ) can also be implemented with minimal overhead. Thus, we do not expect a measurable decrease of performance from our low latency extension. The configured $weight$ guarantees a worst case share of high priority traffic in the maximum throughput TP_{max} of a software router of at least $\frac{TP_{max} \times weight}{\#cores \times \sum_{weights}}$ in case of skewed distribution of high priority traffic and $\frac{TP_{max} \times weight}{\sum_{weights}}$ if we assume high priority traffic that is evenly distributed to all cores.

V. Modeling Software Routers

In this section, we investigate how the performance of off-the-shelf software routers can be improved with respect to low latency traffic treatment. Therefore, we introduce a model of a QoS-aware software router as it is proposed in Section IV. This model is derived from our general modeling approach for resource management in resource-constrained nodes which was published in [41].

As it was already shown by us [24] and other researchers [2], the CPU cores represent the main performance bottleneck of a multi-core software router based on commodity hardware. Therefore, the cores' efficiency constitutes the main performance limiting factor and therefore has to be taken into account in great detail when evaluating such a system.

Each CPU core has to handle one or multiple Rx rings. In case of multiple Rx rings per core, the rings are served in a round robin manner in the standard Linux networking stack. This means that there is currently no support for prioritized packet processing for specific traffic (e.g. real-time traffic) before reaching the CPU bottleneck. However, this is important for software routers in high load situations during which the CPU resources are heavily utilized (cf. Section III).

This shortcoming motivated us to extend our software router model for prioritized packet processing by introducing dedicated Rx rings for the corresponding traffic which will be served according to a specific scheduling strategy (cf. Section V-C). For instance, this extension can be applied to process those packets faster which have low latency constraints like real-time traffic (e.g. VoIP, video conferencing, online gaming).

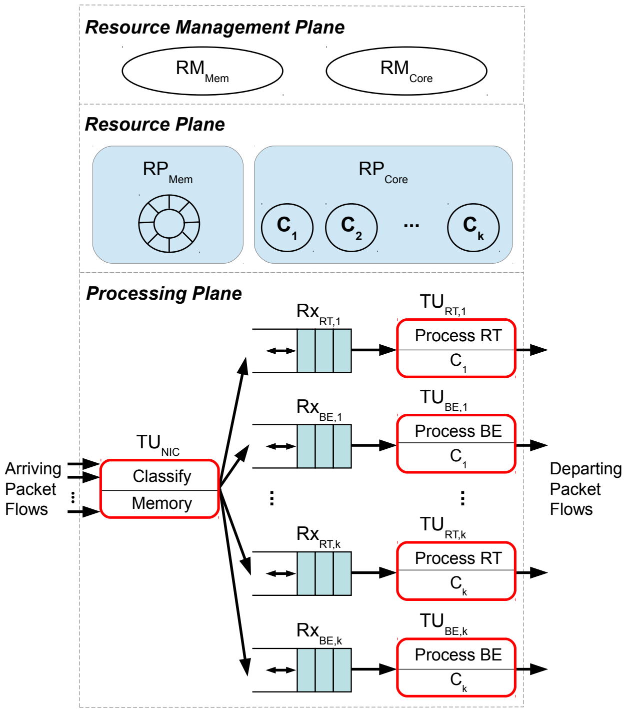


Figure 3. Intra-node resource management model of a software router with low latency support

A. Model of a Software Router with Low Latency Support

Based on a state-of-the-art software router (cf. Section III) and our proposals for a QoS-aware software router (cf. Section IV), we derive a model of a QoS-aware software router which is depicted in Fig. 3. According to our general modeling approach [41], this software router model consists of three planes: the processing plane, the resource plane, and the resource management plane.

1) Processing Plane: In the processing plane, the actual packet processing is modeled based on the task units $TU_{NIC}, TU_{RT,1}, \ldots, TU_{RT,k}$, and $TU_{BE,1}, \ldots, TU_{BE,k}$. The TU_{NIC} abstracts NIC functionalities like traffic classification. The task units $TU_{RT,1}, \ldots, TU_{RT,k}$ respectively $TU_{BE,1}, \ldots, TU_{BE,k}$ represent processes or threads in the operating system to model network stack functionalities (e.g. IP routing table lookup) of real-time respectively best-effort traffic. Furthermore, the Rx rings for real-time respectively best-effort traffic are represented as the task unit queues $Rx_{RT,k}$ respectively $Rx_{BE,k}$ which are located in the resource pool RP_{Mem}. Each task unit queue can store up to 512 packets which refers to the default Rx ring size in a Linux system. Thus, a task unit queue can only store a limited number of packets.

2) Resource Plane: The limited resources of the software router are modeled as resource objects which are located in specific resource pools in the resource plane. For instance, the resource pool RP_{Core} of the CPU cores contains k CPU core resources $C_1, \ldots, C_k$.

3) Resource Management Plane: The resource pools are administered by the (local) resource managers RM_{Core} and RM_{Mem}, whereas a global resource manager as introduced in our previous work [41] can be omitted because there are no dependencies between the two resource types.

When a packet is received, the TU_{NIC} distributes the incoming packet based on specific packet attributes (e.g. TCP/UDP port). Each packet belongs to a specific packet flow which is characterized by a source IP address, a destination IP address, a source port, and a destination port. Besides, the TU_{NIC} classifies packets based on their TOS field (Type of Service) as real-time or best effort traffic. Based on that, the TU_{NIC} maps each flow to a specific task unit queue (Rx ring) of a task unit. In consequence, every packet of a specific flow is served by the same CPU core which is strongly recommended for parallel packet processing on multi-core CPUs to avoid context switches and caches misses [2].

For each packet the TU_{NIC} requires a memory slot in the corresponding Rx ring which is located in the main memory. Thus, the TU_{NIC} has to request the corresponding RM_{Mem} to get an available memory slot from the resource pool RP_{Mem}. Otherwise, if no memory slot is available, then a packet must be dropped.

The replicated task units model the actual parallel packet processing (e.g. IP table lookup, firewall) in a multi-core software router. To process a packet, a task unit requires a resource of the type CPU core. Therefore, it has to request the RM_{Core} for allocating a core resource.

For instance, task unit $TU_{RT,1}$ processes real-time packets and has a higher task unit priority than $TU_{BE,1}$ which processes best effort packets. Thus, the processing of real-time packets is prioritized. Both task units share the same CPU core resource C_1 from the RP_{Core}. Therefore, if $TU_{RT,1}$ needs to process a packet, it is possible that C_1 is currently not available due to its allocation to $TU_{BE,1}$. In this case, resource contention occurs and the RM_{Core} has to arbiter the allocation of C_1 between the task units $TU_{RT,1}$ and $TU_{BE,1}$ based on a resource scheduling strategy (e.g. WFQ). After having been allocated the shared core resource, $TU_{RT,1}$ is able to process packets from its incoming queue $Rx_{RT,1}$. The task unit functionality *Process RT* respectively *Process BE* consumes simulated time corresponding to the required service time depending on the packet size and type of packet processing (e.g. IP forwarding, IPsec encryption, etc.).

B. Model Calibration

The packet latency represents the delay of a packet during its traversal through the software router. It consists of waiting and service times in several system internal components, where it is dominated by the waiting and service time (aka. processing time) at the bottleneck component, i.e. here the set of CPU cores. The waiting time of a packet depends on the number of packets prior to that packet in the task unit queue where the service time depends on the type of packet processing (e.g. IPsec, Routing, Firewall) and its packet attributes (e.g. packet size, real-time or best effort traffic). We focus on the most practice-relevant type of packet processing, namely IP forwarding. In this case, the router's functionality can be classified into two main tasks: (1) packet switching from the incoming port to the outgoing port, and (2) packet processing, like IP routing table lookup or traffic classification.

On the one hand, packet switching is usually packet size dependent due to the fact that most software routers operate on the store-and-forward paradigm while handling packets. According to [53], we apply a simple linear model which consists of a variable part a per Byte (which is dependent on the packet size S) and a constant part b, corresponding to $x = a \cdot S + b$. Based on real testbed measurements (cf. Table 3 of [2]), we derive the calibration values $a \approx 2.34\,\frac{\text{ns}}{\text{B}}$ and $b \approx 272.47\,\text{ns}$ for packet switching (aka. minimal forwarding).

On the other hand, in case of IP forwarding, the effort for packet processing (e.g. updating the IP header) is independent of the packet sizes [53]. Thus, we model the effort for IP packet processing as an additional constant per-packet overhead c. We also derive the calibration value of $c \approx 225.18\,\text{ns}$ from Table 3 of [2]. Consequently, we model the per-packet service time x for IP forwarding as $x = a \cdot S + b + c$.

We also have to estimate the additional latencies induced by other system internal components. In many NICs, *batch processing* reduces the per-packet bookkeeping overhead through handling packets in a "bulk". According to Dobrescu et al. [2] and Kim et al. [54], batching and DMA transfer times increase the packet latency. Based on the work of Dobrescu et al. [2], we estimate that batching introduces a delay for up to 16 packets before DMA transmission which implies 8 packets on average (if we assume uniformly distributed load). Furthermore, the processing of a packet in total requires four DMA transfers: Two transfers from the NIC to the memory (one for the packet and one for its descriptor) and vice versa. We estimate a DMA transfer at $T_{DMA} = 2.56\,\mu\text{s}$. Besides they assume that batching from and to the NIC adds $T_{NIC} = 2 \times 8 \times x$ where x represents the service time in the core. Thus, we estimate an additional packet latency from other non-bottleneck components with $T^{+} = 4 \times T_{DMA} + T_{NIC}$.

In the case of routing a 1518 B packet, the service time is $x = 4.04\,\mu\text{s}$. Then we assume that batching from and to the NIC adds 64.64 μs (2×8×4.04 μs) on average. Based on that, we estimate an additional packet latency from other non-bottleneck components at 74.88 μs (4×2.56 μs + 64.64 μs). This means that at offered loads below the maximum throughput of ca. 1 Mpps (1518 B packet size, 4 CPU cores) the packet latency refers to the sojourn time at the core bottleneck plus 74.88 μs to take into account the additional latency resulting from other system internal components.

C. Resource Management Scheduling Strategies

Each task unit possesses a *task unit priority* TUP_i, $i \in \{1, 2, ..., n\}$ which is used by a resource manager to arbiter between task units which compete for the same shared resource(s). Corresponding to the resource management scheduling strategy, the resource manager prefers a task unit with a high priority where TUP_1 is the lowest priority. Therefore, in case of resource contention, the resource manager may revoke a shared resource from a task unit with low priority based on a specific resource management strategy because another task unit with higher priority is requesting the resource at the same time. We investigate the following standard resource management strategies.

- *First-Come First-Served (FCFS)*: The task unit which allocates the resource first, keeps the resource until it has no further packets to process.
- *Priority (Prio)*: The task unit with the highest task unit priority gets the resource(s) immediately. This strategy is non-preemptive which implies that a task unit is not interrupted during the processing of the current packet.
- *Round Robin (RR)*: The task unit gets the resource(s) for a constant time slice Δt. The respective task unit priorities are not considered, and thus, the time slice size is constant for all task units. The task units are served one after the other (fair queueing). Hence, no starvation occurs.
- *Weighted Fair Queueing (WFQ)*: The task unit gets the resource(s) for a time slice $\Delta t_k = \Delta t \cdot \frac{TUP_k}{\sum_{i=1}^{n} TUP_i}$ corresponding to its task unit priority TUP_k. This implies that the time slice Δt_{high} of a high priority task unit is longer than the time slice Δt_{low} of a task unit with lower task unit priority. Similar to RR, the task units are served one after the other.

In case of the RR and WFQ strategy, we observed that real-time packets incurred strong latency because in practice real-time packets are relatively seldom and thus often have to wait at least a complete time slice to get processed. Therefore, we extend the RR and WFQ strategy to *Low Latency Round Robin (LL-RR)* and *Low Latency Weighted Fair Queueing (LL-WFQ)* for improving low latency support.

LL-RR and LL-WFQ immediately prefer infrequently appearing low latency packets which are processed by a high priority task unit if previously no resource contention occurred. In case of contention, a task unit gets the resource(s) corresponding to the standard behaviour of RR respectively WFQ.

VI. Case Study: Low Latency Packet Processing in Software Routers

In this section, we evaluate and optimize the packet processing performance of an off-the-shelf quad-core software router with respect to low latency packet processing. Therefore, we aim to find optimal resource management scheduling strategies for software routers based on our software router model which was introduced in Section V.

A. Simulation Scenario

The ns-3 simulation scenario consists of multiple load generators and sinks acting as end systems and a router serving as device under test (Fig. 4). The load generators and the sinks have no resource constraints, but the router possesses limited resources, namely 4 CPU cores, and a limited Rx ring size of 512 packets per Rx ring. Our ns-3 resource management extension is applied to model the router under test.

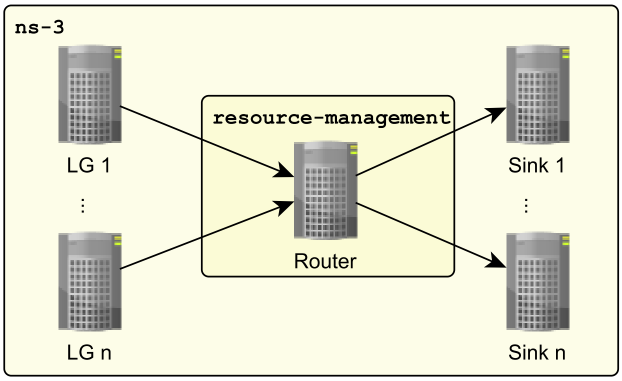


Figure 4. Case study simulation scenario

The load generators and the sinks are connected via 10 GbE point-to-point links to the router which are consciously chosen as high-speed data rates to ensure that the links themselves will not become the bottleneck when applying data transmissions at a high level of offered load. The offered load is a composition of real-time and best effort packet flows corresponding to the mixing proportion (e.g. 30 % real-time traffic). The real-time and best effort traffic is classified by the software router based on the TOS (Type of Service) field of the IPv4 header[1]. We simulate uni-directional traffic from the load generators to the sinks. A packet flow is modeled as a Poisson stream representing a video conferencing session which requires ca. 5 Mbit/s respectively ca. 410 packets/s. The frame size is constant for all traffic corresponding to the MTU of 1518 B of the Ethernet protocol.

According to the CPU utilization of the software router, the video conference packet flows may overload the CPU of the software router for a short period of time. This leads to an increase of the number of packets in the corresponding Rx rings. In this case, the benefit of our low latency extension for software routers (cf. Section IV) can be demonstrated.

B. Simulation Results

We analyzed the mean packet latency for real-time and best effort traffic of the modeled software router with respect to different scheduling strategies (cf. Figs. 5, 6 and 7). As a point of reference to the state of the art, the default Linux networking stack is also represented in a situation when no resource management strategy (no RM) is applied. In this case, only one Rx ring is mapped to a specific core. Thus, all incoming packets are enqueued in the same Rx ring and served according to FCFS service discipline without any prioritization of real-time packets.

In contrast to the state of the art, our low latency extension for the QoS-aware software router uses a dedicated Rx ring for each traffic class per CPU core. This implies that the real-time packets and the best effort packets are enqueued into two separate Rx rings[2] as it is illustrated by Fig. 3.

1) Real-Time Percentage: The real-time percentage is a mixing proportion between real-time and best effort traffic. For instance, a value of 10 % means that on average every tenth packet is a real-time packet and all other packets are best effort packets. Figs. 5(a) and 5(b) show the percentage of real-time traffic of the total traffic on the x-axis and the mean packet latency in microseconds on the y-axis, stated with 95 % confidence intervals. Each of the lines represents a different scheduling strategy with respect to real-time or best effort traffic. The router utilization of 80 % as well as the time slice sizes for RR, WFQ, LL-RR, and LL-WFQ are kept constant in all experiments ($\Delta t = 50\,\mu$s).

When no resource management strategy (no RM) is used, the real-time and best effort packets incur the same mean latency. This case represents a reference to the state-of-the-art as described above.

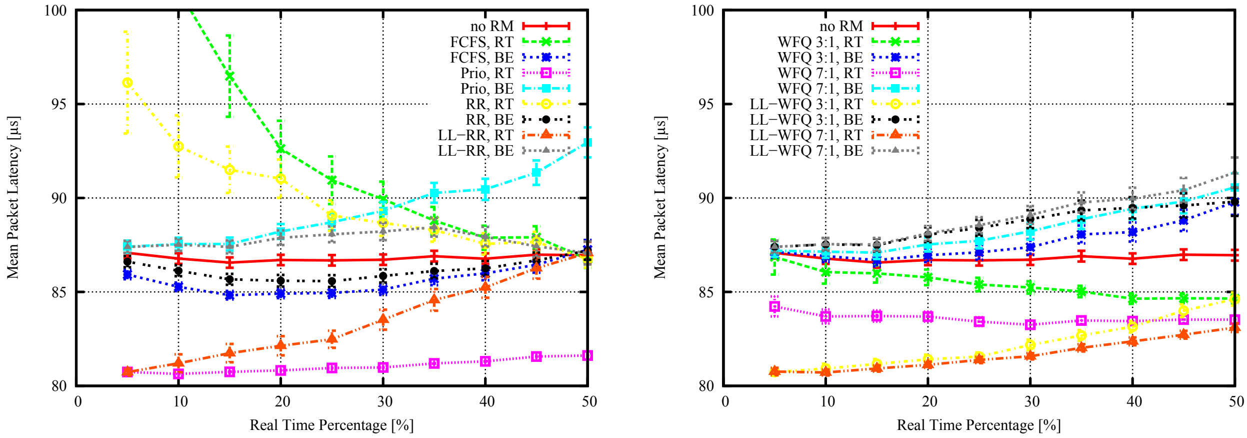
In the case of the FCFS strategy, the less the amount of real-time traffic is the more the real-time traffic suffers from an increase of the mean packet latency because real-time packets often have to wait until all best effort packets are processed. This shows that FCFS is not useful for low latency packet processing.

When applying the Prio strategy, an incoming real-time packet (which is processed by a high priority task unit) is always served before a best effort packet. Thus, with the Prio strategy real-time packets are served faster at the expense of the best effort packets. This effect is more dominant for lower percentages of real-time traffic. With higher percentages of real-time traffic, there are many real-time packets in the same Rx ring which are served in a FCFS manner. Thus, the mean packet latency of real-time packets increases. If there is no real-time traffic (real-time percentage is 0 %) then the mean packet latency of best effort packets equals the latency with state-of-the-art processing, when no scheduling strategy is applied. The Prio strategy represents borderline cases with respect to the mean packet latency for all scheduling strategies, except FCFS and RR. It is the lower-bound for the real-time packets whereas it is an upper-bound for the best effort packets.

The RR strategy yields to an increase of the mean packet latency for real-time traffic (whenever the real-time percentage is small) because real-time packets often have to wait for one complete time slice to get processed.

[1] The distinction of real-time and best effort traffic can also be done based on any other packet attribute (e.g. TCP/UDP port) which is supported by the NIC controller.

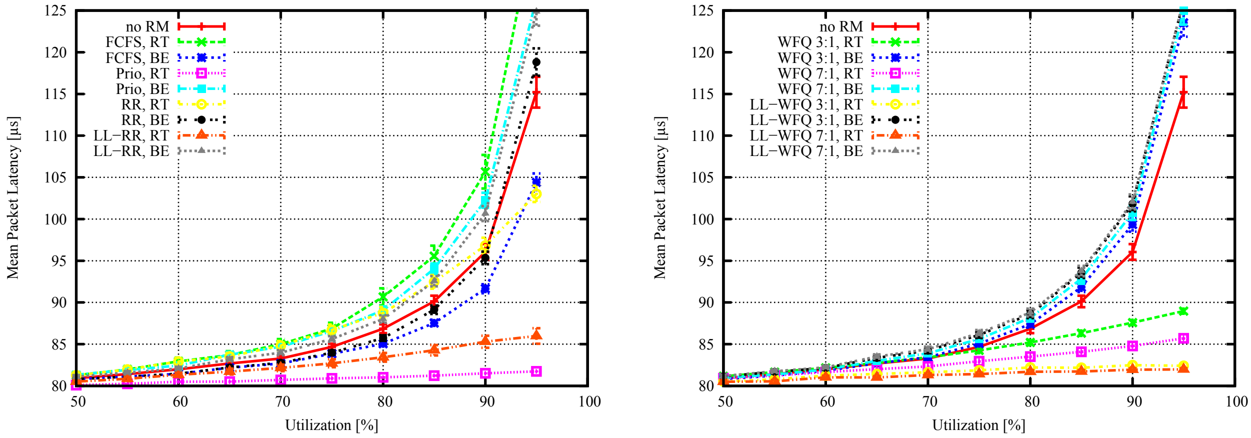
[2] In this case study, we only define two traffic classes but also multiple traffic classes can be used which are mapped to multiple Rx rings.



(a) FCFS, Prio, RR, and LL-RR

(b) WFQ and LL-WFQ

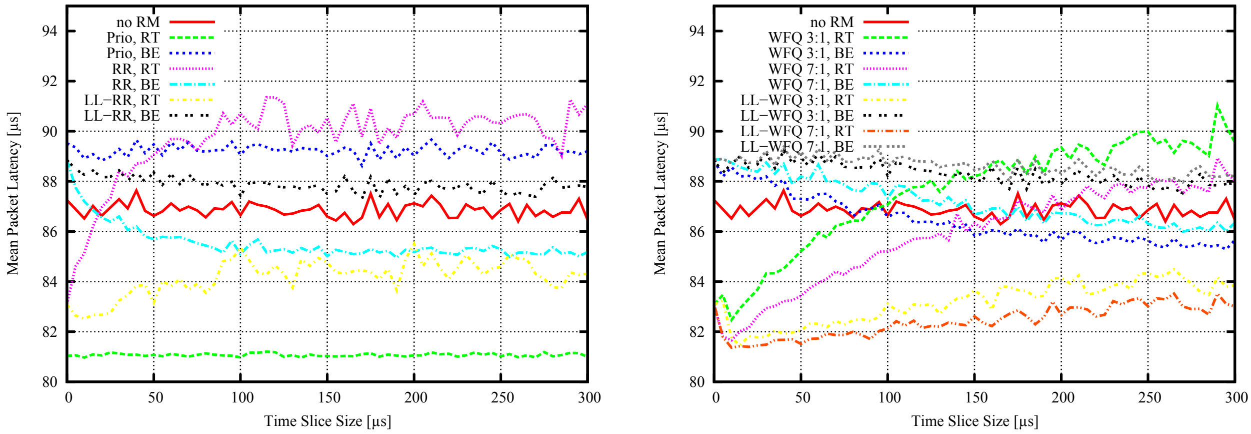
Figure 5. Mean packet latency as function of real-time percentage for different resource management scheduling strategies



(a) FCFS, Prio, RR, and LL-RR

(b) WFQ and LL-WFQ

Figure 6. Mean packet latency as function of utilization for different resource management scheduling strategies



(a) Prio, RR, and LL-RR

(b) WFQ and LL-WFQ

Figure 7. Mean packet latency as function of the time slice size for different resource management scheduling strategies

This implies that RR is also not applicable for low latency packet processing. However, this behavior can be mitigated with the help of our low latency extension (cf. Section V-C) which is applied in the LL-RR strategy. With the LL-RR strategy, the mean packet latency of real-time packets becomes smaller the less the percentage of the real-time traffic is because the time slice sizes for both traffic classes are equal and the more packets must be handled per time slice the longer is the waiting time. Besides, LL-RR becomes similar to RR at high percentages of real-time traffic because it is more likely that resource contention occurs and each task unit gets its corresponding time slice. When the percentage of real-time traffic is 50 %, the real-time and best effort traffic suffer the same mean packet latency.

Fig. 5(b) illustrates the same measurements for WFQ and LL-WFQ where the relations of the time slice sizes between the real-time and best effort task unit are denoted. For instance, a relation 3:1 means that the time slice size of the real-time task unit is three times larger than the time slice size of the best effort task unit. Like the RR strategy, WFQ also works insufficient with respect to the handling of real-traffic. This behavior can also be improved by extending WFQ to LL-WFQ with the low latency extension. The LL-WFQ strategy shows similar behavior as the Prio strategy at low real-time percentages because in most cases no resource contention occurs. This implies that when a real-time packet is received the real-time packet is immediately processed because the corresponding task unit gets immediately the resource CPU core (e.g. directly after the processing of the current best effort packet).

In summary, the improvement of the mean packet latency of the real-time traffic is achieved at the expense of the best effort traffic with all strategies. For instance, with the Prio strategy and 10 % real-time traffic, a real-time packet requires ca. 81 μs which is 7 % faster than the state of the art (no RM) with ca. 87 μs. This improvement is achieved at the cost of the best effort packets where the mean packet latency is only increased by 1 % to ca. 88 μs.

At a CPU utilization of 80 %, the real-time packets strongly benefit from the introduction of a scheduling strategy like Prio. This effect is strengthened at higher values of utilization. For practical use cases, we assume a real-time percentage from 10 up to 30 %.

2) Utilization: The utilization is a metric for the degree of occupation of a specific resource which is defined as the relation between the busy time of the resource and the total time of observation. In this case, the utilization refers to the bottleneck resources which are here the CPU cores. Figs. 6(a) and 6(b) show the utilization of the software router on the x-axis and the mean packet latency in microseconds on the y-axis. The mean packet latency is stated with 95 % confidence intervals. The real-time percentage of ca. 30 % as well as the time slice sizes for RR, WFQ, LL-RR, and LL-WFQ are kept constant in all experiments ($\Delta t = 50\,\mu$s).

At values less than 50 % utilization, the resource CPU core is often idle. Here, the mean packet latency is nearly equal for all scheduling strategies. For a high-speed software router, we assume utilization values of 50 % and above. When the CPU core is busy, the corresponding task unit (and also the packets) has to wait until the resource becomes available which leads to an increase of the packet latency. If the utilization increases up to 100 % then the mean packet latency increases exponentially up to a maximum which is determined by the Rx ring size. The mean packet latency is no longer well-defined because arriving packets often come up with a full queue (aka. Rx ring) and must be dropped. Hence, the stated mean packet latency refers only to the successfully served packets.

The case for the default behavior when no resource management strategy is applied (no RM) is shown as a reference to the state of the art. The FCFS and RR strategies are not helpful for the real-time packets as already discussed above.

When applying the Prio strategy, the mean packet latency of best effort packets exponentially increases whereas it only linearly rises for real-time packets because the real-time traffic is rather low and the corresponding real-time packets are always served prior to the best effort traffic. This effect even holds for high values of utilization and is beneficial for resource-constrained nodes to satisfy low latency requirements.

In case of the WFQ, if the utilization increases then the higher the priority of the real-time task unit is (and thus the corresponding time slice size) the less is the mean packet latency of the real-time packets.

With LL-RR and LL-WFQ, the mean packet latency of real-time traffic just linearly increases if the utilization increases and is close to the Prio strategy for low values of utilization because it is likely that no resource contention occurs. However, if the utilization increases then in most cases there is resource contention which causes a rise of the mean packet latency of real-time traffic.

3) Time Slice Size: The time slice, or also called *quantum*, is defined as the period of time for which a process (or task unit) is allowed to run in a preemptive multitasking system. Figs. 7(a) and 7(b) show the time slice size on the x-axis and the mean packet latency in microseconds on the y-axis. The confidence intervals are omitted for better readability. Each of the lines represents a different scheduling strategy with respect to real-time or best effort traffic. The real-time percentage of 30 % as well as the router utilization of 80 % are kept constant in all experiments.

The time slice size is only relevant for the scheduling strategies RR, LL-RR, WFQ, and LL-WFQ. Nonetheless, the case for “no resource management” is also depicted as a point of reference to the state of the art. Again, the Prio strategy represents borderline cases with respect to the mean packet latency for LL-RR, WFQ and LL-WFQ, except RR. It represents a lower-bound for the real-time packets whereas it depicts an upper-bound for the best effort packets.

In case of the RR strategy and small time slices, the relatively infrequent real-time packets benefit from often getting the resource. When the time slice size increases, then the mean latency of real-time traffic significantly increases because it is likely that real-time traffic can only be served if the time slice is over or all best effort packets were processed. This unfavorable behavior of RR can also be mitigated with our low latency extension (cf. Section V-C).

When applying the WFQ strategy, we observe a behavior similar to RR but the increase of the mean packet latency is weaker than with RR if the time slice size increases. This increase becomes weaker the higher the priority of the real-time task unit is (cf. WFQ 3:1 and WFQ 7:1). Again, this behavior is eliminated with LL-WFQ (as with LL-RR). Therefore, with LL-RR and LL-WFQ, the mean packet latency of real-time traffic is always better than the state of the art independently of the time slice size.

VII. Conclusion

In this paper we described how PC-based multi-core packet processing systems can be optimized for low latency packet processing. We proposed a QoS-aware software router architecture where the incoming packets are classified by the NIC into dedicated Rx rings for real-time and best-effort traffic. In combination with a scheduling strategy, specific packet flows can be prioritized before reaching the CPU bottleneck. Therefore, our approach is in contrast to classical QoS techniques such as DiffServ and IntServ which assume that the outgoing link is the bottleneck. This enhancement has the focus on but is not restricted to software routers. Our architecture just utilizes technology that is already available in commodity servers today. We used our approach for modeling of resource contention in resource-constrained nodes which is also implemented as the *resource-management* extension module for ns-3. Based on that, we derived a QoS-aware software router model which we used to optimize the performance of a software router with respect to low latency packet processing. Our case studies showed that the scheduling strategy of a software router has significant influence on the performance of handling real-time traffic.

In future research, we plan to carry out more fine-grained testbed measurements to refine our resource-constrained software router model in terms of further performance-relevant details. For instance, a more accurate modeling of the effects on the intra-node latency resulting from other system internal components (e.g. driver) will be one of the next steps. Furthermore, we plan to conduct testbed measurements with a prototype of the described QoS-aware software router. Moreover, we will investigate further resource management scheduling strategies with the help of our ns-3 extension. Finally, we hope to be able to identify further performance-limiting factors and bottlenecks of existing software routers as well as to predict effects caused by changes and optimizations in the router software.

Acknowledgments

We would like to acknowledge the valuable contributions through numerous in-depth discussions from our colleagues Dr. Klaus-Dieter Heidtmann, Andrey Kolesnikov, Alexander Beifuß, Paul Emmerich, Dominik Scholz, Stefan Edinger, and Paul Lindt.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, “B4: Experience with a Globally-deployed Software Defined WAN,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, August 2013.

[2] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, “RouteBricks: Exploiting Parallelism To Scale Software Routers,” in *ACM Symposium on Operating Systems Principles (SOSP)*, October 2009.

[3] B. Munch, “Hype Cycle for Networking and Communications,” Gartner,” Report, July 2013.

[4] R. Bolla and R. Bruschi, “PC-based Software Routers: High Performance and Application Service Support,” in *ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, August 2008, pp. 27–32.

[5] M. Dobrescu, K. Argyraki, and S. Ratnasamy, “Toward Predictable Performance in Software Packet-Processing Platforms,” in *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2012.

[6] L. Rizzo, “Netmap: A Novel Framework for Fast Packet I/O,” in *USENIX Annual Technical Conference*, April 2012.

[7] F. Fusco and L. Deri, “High Speed Network Traffic Analysis with Commodity Multi-core Systems,” in *Internet Measurement Conference*, November 2010, pp. 218–224.

[8] *Data Plane Development Kit: Programmer’s Guide, Rev. 6*, Intel Corporation, January 2014.

[9] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: A GPU-Accelerated Software Router,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, August 2011.

[10] M. Hassan, A. Nayandoro, and M. Atiquzzaman, “Internet Telephony: Services, Technical Challenges, and Products,” *IEEE Communications Magazine*, vol. 38, no. 4, pp. 96–103, August 2000.

[11] T. Meyer, D. Raumer, F. Wohlfart, B. E. Wolfinger, and G. Carle, “Low Latency Packet Processing in Software Routers,” in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Best Paper Award*, July 2014.

[12] R. Huggahalli, R. Iyer, and S. Tetrick, “Direct Cache Access for High Bandwidth Network I/O,” *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 50–59, May 2005.

[13] V. Tanyingyong, M. Hidell, and P. Sjodin, “Using Hardware Classification to Improve PC-based OpenFlow Switching,” in *International Conference on High Performance Switching and Routing (HPSR)*, July 2011, pp. 215–221.

[14] *Intel Ethernet Controller X540 Datasheet Rev. 2.7*, Intel Corporation, March 2014.

[15] V. Tanyingyong, M. Hidell, and P. Sjodin, “Improving Performance in a Combined Router/Server,” in *International Conference on High Performance Switching and Routing (HPSR)*, June 2012, pp. 52–58.

[16] A. Tedesco, G. Ventre, L. Angrisani, and L. Peluso, "Measurement of Processing and Queuing Delays Introduced by a Software Router in a Single-Hop Network," in *IEEE Instrumentation and Measurement Technology Conference*, May 2005, pp. 1797–1802.

[17] P. Carlsson, D. Constantinescu, A. Popescu, M. Fiedler, and A. Nilsson, "Delay Performance in IP Routers," in *Performance Modelling and Evaluation of Heterogeneous Networks*, July 2004.

[18] G. Almes, S. Kalidindi, and M. Zekauskas, "A One-way Delay Metric for IPPM," RFC 2679, IETF, September 1999.

[19] A. Botta, A. Dainotti, and A. Pescapé, "Do You Trust Your Software-based Traffic Generator?" *IEEE Communications Magazine*, vol. 48, no. 9, pp. 158–165, 2010.

[20] P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," *ArXiv e-prints*, Oct. 2014.

[21] G. A. Covington, G. Gibb, J. W. Lockwood, and N. Mckeown, "A Packet Generator on the NetFPGA Platform," in *IEEE Symposium on Field Programmable Custom Computing Machines*, 2009, pp. 235–238.

[22] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation," in *Passive and Active Measurement*. Springer, March 2012, pp. 85–95.

[23] R. Bolla and R. Bruschi, "Linux Software Router: Data Plane Optimization and Performance Evaluation," *Journal of Networks*, vol. 2, no. 3, pp. 6–17, June 2007.

[24] T. Meyer, F. Wohlfart, D. Raumer, B. E. Wolfinger, and G. Carle, "Validated Model-Based Performance Prediction of Multi-Core Software Routers," *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, vol. 37, no. 2, pp. 93–107, 2014.

[25] T. Begin, A. Brandwajn, B. Baynat, B. E. Wolfinger, and S. Fdida, "High-level Approach to Modeling of Observed System Behavior," *Perform. Eval.*, vol. 67, no. 5, pp. 386–405, May 2010.

[26] K. Salah, "Modeling and Analysis of PC-based Software Routers," *Computer Communications*, vol. 33, no. 12, pp. 1462–1470, 2010.

[27] R. Chertov, S. Fahmy, and N. Shroff, "A Device-Independent Router Model," in *IEEE Conference on Computer Communications (INFOCOM)*, April 2008.

[28] J. Pan and R. Jain, "A Survey of Network Simulation Tools: Current Status and Future Developments," Washington University in St. Louis, Tech. Rep., November 2008.

[29] E. Weingärtner, H. vom Lehn, and K. Wehrle, "A Performance Comparison of Recent Network Simulators," in *IEEE International Conference on Communications (ICC)*, June 2009, pp. 1–5.

[30] S. McCanne, S. Floyd, K. Fall, K. Varadhan, *et al.*, "Network Simulator ns-2," http://www.isi.edu/nsnam/ns, 1997.

[31] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network Simulations with the ns-3 Simulator," *ACM SIGCOMM Demonstration*, August 2008.

[32] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 35–59.

[33] "OPNET Application and Network Performance," www.opnet.com, November 2014.

[34] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[35] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs," in *ACM Design Automation Conference*, 2011, pp. 1050–1055.

[36] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2011, pp. 1–12.

[37] P. Suresh and R. Merz, "NS-3-Click: Click Modular Router Integration for NS-3," in *International Conference on Simulation Tools and Techniques (ICST)*, March 2011.

[38] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, August 2000.

[39] Q. Wu and T. Wolf, "Runtime Task Allocation in Multicore Packet Processing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 10, pp. 1934–1943, October 2012.

[40] S. Kristiansen, T. Plagemann, and V. Goebel, "Modeling Communication Software Execution for Accurate Simulation of Distributed Systems," in *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, May 2013, pp. 67–78.

[41] T. Meyer, B. E. Wolfinger, S. Heckmüller, and A. Abdollahpouri, "Extensible and Realistic Modeling of Resource Contention in Resource-Constrained Nodes," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Best Paper Award*, July 2013.

[42] S. Abbas, M. Mosbah, and A. Zemmari, "ITU-T Recommendation G.114, One Way Transmission Time," *Lect. Notes in Comp. Sciences*, May 2007.

[43] *Intel 82599 10 Gigabit Ethernet Controller Datasheet Rev. 2.76*, Intel Corporation, October 2012.

[44] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond Softnet," in *5th Annual Linux Showcase & Conference*, vol. 5, 2001, pp. 18–18.

[45] J. H. Salim, "When NAPI comes to town," in *Linux Conference*, 2005.

[46] "eXtensible Open Router Platform," http://www.xorp.org/, November 2014.

[47] "Quagga Routing Suite," http://www.nongnu.org/quagga/, November 2014.

[48] A. Beifuß, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle, "A Study of Networking Software Induced Latency," in *2nd International Conference on Networked Systems (NetSys)*, March 2015.

[49] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *RFC 2475*, 1998.

[50] J. Wroclawski, "The Use of RSVP with IETF Integrated Services," *RFC 2210*, 1997.

[51] "PF_RING High-Speed Packet Capture, Filtering and Analysis," http://www.ntop.org/products/pf_ring/libzero-for-dna, November 2014.

[52] T. Herbert and W. de Bruijn, "Scaling in the Linux Networking Stack," https://www.kernel.org/doc/Documentation/networking/scaling.txt, November 2014.

[53] R. Ramaswamy, N. Weng, and T. Wolf, "Analysis of Network Processing Workloads," *Journal of Systems Architecture*, vol. 55, no. 10, pp. 421–433, December 2009.

[54] J. Kim, S. Huh, K. Jang, K. Park, and S. Moon, "The Power of Batching in the Click Modular Router," in *Asia-Pacific Workshop on Systems*, July 2012, p. 14.

Torsten M. Runge received his M.Sc. degree in Information Technology/Computer Science from the University of Rostock in 2006. He worked for Siemens and Deutsche Telekom (2007-2011) where he participated in several international research projects dealing with routing in wireless mesh and sensor networks. Since 2012 he has been engaged in research projects at the University of Hamburg. His research interests include computer networks with particular emphasis on routing as well as parallel packet processing.

Daniel Raumer is research associate at the chair for Network Architectures at the Technische Universität München (TUM), Germany, where he received his B.Sc. and M.Sc. in Informatics, in 2010 and 2012. He is concerned with device performance measurements with relevance to Network Function Virtualization as part of Software-defined Networking architectures.

Florian Wohlfart is a Ph.D. candidate working at the chair for Network Architectures and Services at Technische Universität München. He received his M.Sc. in computer science at Technische Universität München in 2012. His research interests include software packet processing, middlebox analysis and network performance measurements.

Bernd E. Wolfinger received his Ph.D. in Computer Science (Dr. rer. nat.) from University of Karlsruhe in 1979. From 1975 until 1980 he was a member of the scientific staff at Nuclear Research Center, Karlsruhe and became an Assistant Professor at University of Karlsruhe in 1981. Since October 1981 he has been a Professor of Computer Science at University of Hamburg, where he is currently heading the Telecommunications and Computer Networks (TKRN) Subdivision. He has been editor of books and special issues of journals; he has published more than 150 papers in areas such as high-speed & mobile networks and real-time (audio/video) communications as well as in modeling (simulation & queuing networks), measurement, traffic engineering, performance & reliability evaluation, QoS management and e-learning. Prof. Wolfinger has been a member of IEEE and of Gesellschaft für Informatik (GI) as well as a Senior Member of ACM.

Georg Carle is professor at the Department of Informatics of Technische Universität München, holding the chair for Network Architectures and Services. He studied at University of Stuttgart, Brunel University, London, and Ecole Nationale Superieure des Telecommunications, Paris. He did his PhD in Computer Science at University of Karlsruhe, and worked as postdoctoral scientist at Institut Eurecom, Sophia Antipolis, France, at the Fraunhofer Institute for Open Communication Systems, Berlin, and as professor at University of Tübingen.