Torsten Meyer*, Florian Wohlfart, Daniel Raumer, Bernd E. Wolfinger and Georg Carle

Validated Model-Based Performance Prediction of Multi-Core Software Routers

Abstract: The possibility of using flexible and cost-efficient commodity hardware instead of expensive custom hardware has triggered wide interest in software routers. Performance measurement and simulation are important approaches for identifying bottlenecks of such systems to predict and improve the performance. We measure the performance of software routers using current multi-core hardware architectures. We introduce an innovative and validated node model for intra-node resource contention and realized a resource management extension for the widely-used network simulator ns-3 which allows to evaluate and predict the performance of current and future software router architectures.

Index Terms: measurement, simulation, intra-node model, resource contention, model validation, software router, performance prediction

*Torsten Meyer: E-Mail: meyer@informatik.uni-hamburg.de
Florian Wohlfart: E-Mail: wohlfart@net.in.tum.de
Daniel Raumer: E-Mail: raumer@net.in.tum.de
Bernd E. Wolfinger: E-Mail: wolfinger@informatik.uni-hamburg.de
Georg Carle: E-Mail: carle@net.in.tum.de

I Introduction

The performance of commodity PC hardware has increased rapidly. Therefore, it becomes feasible to use off-the-shelf systems as servers or routers. For instance all Unix-based systems are capable of basic routing functionality. Every PC can be transformed into a software router with the aid of special software or by selecting an appropriate operating system (OS). Thereby commodity hardware is more cost-efficient than specialized hardware solutions and network components. Leveraged by high flexibility and low costs of software developments in comparison with hardware developments, software solutions are preferred in many scenarios. While this advantage may be counterbalanced by the higher performance and lower energy consumption of specialized hardware, the arguments of higher flexibility and better cost-efficiency still remain.

Software routers allow rapid deployment of new features that require a considerably more expensive and time-consuming development cycle when implemented by dedicated hardware. For instance, the IETF NETCONF WG or the Open Networking Foundation implemented new approaches for the configuration of network devices in software, whereas hardware implementations were not available in the early stage.

Nevertheless the benefits of software routers come with the drawback of smaller number of ports and lower performance. Approaches like RouteFlow [1] combine the flexibility and routing functionality of software routers with the forwarding performance of hardware OpenFlow switches. Thereby the best of both worlds can be achieved in one system: a comparatively cheap routing system with the performance and scalability of dedicated networking hardware that has the flexibility of software routers. Researches have shown that the performance of specialized routing hardware is within the reach of software routers [2], [3]. For the use of computer systems in high-speed networks the traffic of 1 Gbps, 10 Gbps, and higher speed networks has to be managed. Systems have to be able to cope and handle this traffic without loss of data. Bus systems must guarantee the required data rates between the hardware components. Depending on the type of routing tasks, the OS has to do diverse complex treatments per packet. In order to improve the performance it is necessary to understand packet processing limitations in PC systems in detail. Achieved performance gains can indeed be explained qualitatively on the basis of the hardware architecture and the processes in the OS, but there is a lack of models that could explain the results quantitatively or even predict them. A good model provides a simple way to gain insight into complex packet processing tasks. This model should be scalable and applicable for manifold simulation scenarios which cannot be provided in real testbeds.

In this paper, we measure and simulate the performance of software routers based on current multi-core systems. Hence, we apply our general concept for realistic modeling of resource contention in resource-constrained nodes. Our modeling approach is implemented as a resource management module for the network simulator ns-3. After calibrating and validating our model based on our testbed measurements, we evaluate and predict the performance of existing and future software router architectures. This paper is an extended and revised version of [4].

The remainder of the paper is organized as follows. Section II outlines the state of the art in modeling, measuring and implementation of software routers. Section III shows the characteristics and the components which are needed to setup a software router. In Section IV we introduce our general model for simulation of intra-node resource contention. Section V describes our testbed which was used for the calibration of our model. Section VI presents a case study to compare the real testbed measurements with our simulation results to validate our modeling approach. Finally, we summarize our contribution and give an outlook in Section VII.

II Related Work

Scientists aiming for low level optimizations [5]–[7] need detailed knowledge of the complex system of a software router. The interactions between the kernel and drivers, but also of application layer above and the underlying layer of hardware must be analyzed. In this case, modeling and simulation techniques [8]–[12] can help to understand the related effects and performance factors.

Some projects consider surveying software router implementations as a task with the goal of providing hints for future optimizations [2], [3], [13]. In context of the Route-Bricks project [2], the authors analyzed the performance influences of multi-core PC systems. They parallelized packet processing on multi-core CPUs and extended this approach to a cluster of software routers. PacketShader [5] utilizes the GPU to speed up packet processing. PF_RING [6] and netmap [7] focus on the utilization of DMA features in order to avoid copy operations that are normally needed to get packets to the user space.

In modeling and simulation of such complex systems several approaches were proposed. Chertov et al. [8] introduced a device-independent router model which just considers the queue size and number of service units inside a router. Thus the model can be used for different router types by tuning specific parameters. Bobrek et al. [9] used a hybrid simulation/analytical approach for modeling shared resource contention in Programmable Heterogeneous Multiprocessor (PHM) systems. Besides, Sokolsky [10] followed a formal approach to model resource constraints in real-time embedded systems which is easily extensible to include new kinds of resources and resource constraints. Begin et al. [11] proposed a high-level approach for modeling an observed system behavior with little knowledge of the system internal structure or operation. This is done by adequately selecting the parameters of a set of queueing systems and queueing networks. Bjork-

man and Gunningberg [12] investigated the effects of locks and memory contention which are major performance bottlenecks in multi-processor systems. They presented a queueing network model for performance prediction of a shared memory multi-processor with parallel protocol processing.

Measuring of network devices in general was standardized by the IETF in RFC 2544 [14]. Bolla and Bruschi [13] applied RFC 2544 for software router performance. Beside external measuring via dedicated hardware they refined an internal view on packet processing in Linux 2.6 via profiling and the knowledge about hardware architecture at this time. Dobrescu et al. [3] published a study on the predictability of software networking equipment.

Beside measuring of software routers in whole or facets, other projects aim for the implementation of software routers. These routers are also referred to as open routers (OR) to clarify the contrast to the relatively inflexible closed source hardware routers. XORP [15], Quagga [16] and BIRD [17] are the most well-known ORs. The Vyatta Open-FirmwareRouter [18] is a Debian based Linux distribution equipped with network applications for routing like Quagga and OpenVPN. Vyatta's business model demonstrates the marketability of software routers as it is based on deployment, support and training for their software router distribution. Therefore it includes other stakeholders besides the scientific community. In contrast the Click Modular Router [19] was used mainly for scientific research. Other examples for PC-based packet processing are ServerSwitch [20], as proof of concept for flexible packet switching in cloud data centers, and Open vSwitch [21] which is a software switch also used as a reference implementation of OpenFlow. Open vSwitch has been ported to multiple hardware platforms and made part of the Linux kernel.

III Realization of a Software Router

The IP protocol was designed to provide a best-effort service to the transport layer in a decentralized way. Due to the decentralization each IP router decides on its own to which neighbor it has to forward an incoming packet. Therefore, an IP router must keep a state in a routing table to track the networks reachable via its neighbors. In summary the duties of an IP router are twofold: first, it needs to learn its routing table (either via manually installed rules or a routing protocol) and second, a router needs to forward the actual traffic according to its routing table. The parts of a router concerned with learning and updating the routing table form the *control plane*, while the parts of the

router dedicated to per-packet forwarding are referred to as the *forwarding plane* [22].

The control plane implements various distributed routing protocols, such as RIP, OSPF, or BGP. Using these protocols the router either gains a global or local view of the network topology. From this topology information the control plane derives the routing table. As the processing of the routing protocol messages is rather complicated but less time-critical, the control plane is usually implemented in software, that is running on general-purpose processors, even in dedicated hardware routers [22].

On the other hand, the tasks of the forwarding plane are rather simple but critical in terms of throughput and latency. In addition to the actual forwarding other tasks like routing table lookups, TTL decrements, fragmentation, and checksum recalculation also belong to the forwarding plane. In hardware routers these tasks are implemented using special-purpose chips (ASICs) [22]. This section gives an overview of the routing software and selected optimizations in software and hardware that are relevant for software routers.

A Routing Software

When building a router using commodity hardware, we need to implement both the control plane and the forwarding plane in software. As described above, the control plane is generally implemented in software, mainly because it is not critical for the performance of the router. Therefore, we focus on the forwarding plane, which directly affects routing performance.

As mentioned in Section II, there are several mature implementations of software routers for UNIX-like platforms, such as Linux, FreeBSD, NetBSD, and Solaris. These platforms support the full functionality of the forwarding plane. Therefore, static routing is supported without additional software. When distributed routing protocols have to be used to generate the routing information, extra software is required. The routing software packages Quagga [16], BIRD [17], and the eXtensible Open Router Platform (XORP) [15] support the most common routing protocols, such as RIP, OSPF and BGP. While Quagga and BIRD only provide control plane functionality, XORP also allows to change the forwarding plane of the OS. XORP relies on the Click implementation of the forwarding plane, which we will discuss in the next section.

B Software-Based Routing Performance Optimizations

The functionality of the forwarding plane in UNIX-like systems by default uses the general-purpose network stack. Due to its general use it is not explicitly optimized for high-performance packet forwarding. The Click Modular Router [19] provides a replacement for the OS network stack with its own implementation of the forwarding plane. In contrast to the software routers discussed in the last section, which come as ready-to-use packages, Click only provides a framework to build software routers. It offers modules, which provide simple parts of the routing and forwarding functionality, like filters, queues, TTL decrement, or checksum calculation. These modules are connected by a directed graph. Paths in such a graph represent a connection on which a packet can travel from one module to another. Hence, Click is very flexible and allows to build almost any kind of packet processing software – such as an IP router.

Setting up an own Click processing graph is costly, but can provide a better packet forwarding performance than the pure OS network stack. Click was optimized for fast and flexible packet processing, so that it outperforms the Linux network stack [19]. Additionally, Click graphs can be customized and tailored to a certain use case allowing it to perform even better.

The Click community contributed modules and extensions, like the support for netmap [7] or an OpenFlow switch element [23]. Click elements such as a load generator and a load sink contributed by the author of [24] show that it is possible to implement almost any kind of packet processing using Click.

The standard Linux network stack has received various optimizations during the last years, too. These optimizations led to the New API (NAPI). New software techniques introduced with the NAPI, incorporated in kernel 2.6, combine or offload processing steps, avoid interrupts, and avoid memory allocations and copy operations. In experiments we spotted a performance increase of about 7% from Linux kernel 2.6.35.9 to kernel 3.2.39 and even roughly 10% to 3.8.2. Given these performance increases it is even more surprising that older kernel versions are still broadly used. For example Debian "Squeeze", which was replaced by "Wheezy" in May 2013, is shipped with Linux kernel 2.6.32 released in 2009. Debian "Wheezy" – the latest stable release – comes with kernel 3.2, that was released in January 2012.

Up to now, packet processing applications achieve high performance by running in kernel mode and thus being able to access kernel managed buffers without copy-

ing data to user space. A big drawback of this approach is that applications in kernel mode can easily crash the system. While interfaces between driver, OS, and applications have been untouched for years these borders have been exceeded recently. Zerocopy packet processing aims to avoid costly copy operations between DMA accessible buffers, kernel-level buffers, and user space buffers. Prominent examples are PF_RING [6] coming from Linux and netmap [7] from FreeBSD, which was also ported to Linux in 2012. PF_RING modifies drivers in order to let the network interface directly access a ring buffer using DMA. Packets in this ring buffer are mapped into the user space. Conceptional drawbacks result from the fact that applications have to be adapted to this stack and from the introduction of a delay from the moment of the finished DMA copy until the mapping from the ring buffer to the user space happens. Netmap uses a similar approach by mapping the DMA accessed space directly into the user space. A kernel module controls access to the storage used by the different actors which also enforces modifications in drivers. So the performance increases came at the cost of adjusting a stable well known interface and therefore losing some independence from the underlying hardware. Currently the described zero-copy techniques come with modified driver versions of the e1000e, igb, and ixgbe Linux driver for Intel network interfaces.

C Hardware-Based Routing Performance Optimizations

A software router is based on commodity server hardware, which made a steady development during the last years. Within this development process new hardware features lead to the development of new software and the other way round. Optimizations caused by the need for higher inter-component connection speed triggered the change from bridge to hub architectures. The current hub architecture is displayed schematically in Fig. 1 (cf. [25]). Components were integrated with others for sake of communication optimization and the increase of density. The memory controller is placed onchip since Intel's Core i7 (2008) and AMD's K8 (2003) architecture. Therefore, it is referred to as integrated memory controller (IMC). Another trend in hardware architecture is a steady growing degree of parallelization. Intel CPUs and the I/O Hub communicate to each other via QuickPath Interconnect (QPI) [26].

On the other side, offload mechanisms try to shift workload from the CPU to the specialized hardware components and thus discharge the CPU from some of its load. Modern NICs support mechanisms like the TCP Segmenta-

tion Offload (TSO). TSO outsources TCP segmentation of large user data blocks from the CPU to the network interface which reduces the CPU load on the sending side caused by the network protocol stack. The same technique also exists on the receiving side and for UDP. The NIC automatically reassembles received TCP segments again. Direct Memory Access (DMA) is in use for some years, allowing the NIC to access the memory without producing load for the CPU. New network cards already implement the next step called Direct Cache Access (DCA). DCA allows for direct writing into the CPUs cache and therefore avoids several hundred CPU cycles per packet that would be spent with waiting for data otherwise. Beside offload techniques, interrupt moderation or interrupt coalescence are further examples for optimizations. NICs wait for the arrival of more packets, which are then passed to the operating system in a bundle, before triggering an expensive interrupt. The Receive Side Scaling (RSS) technique allows the NIC to enqueue packets according to their flow affiliation to a certain queue. Each queue is connected to another core. So packets of a flow are processed always by the same core. Packets of the same flow are likely to use the same data for forwarding decisions and to access the same state information (if a state is required). Therefore, RSS cares for cache locality and allows for better parallelism.
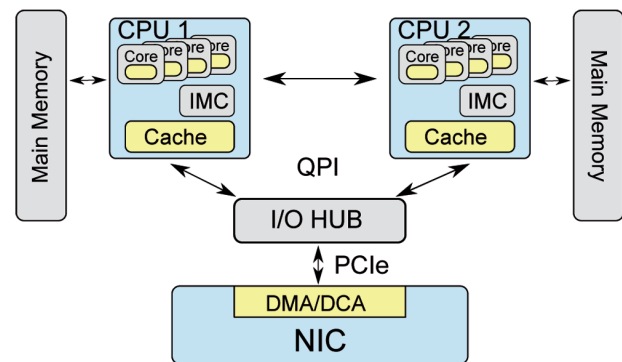

Fig. 1: Intel Hardware Architecture.

IV Performance Evaluation with Simulations

In this section, we give an overview of current network simulators with respect to modeling of intra-node resource contention. Based on that, we introduce our unified model for intra-node resource management in resource-constrained network nodes. We show the most important implementation aspects of our resource management exten-
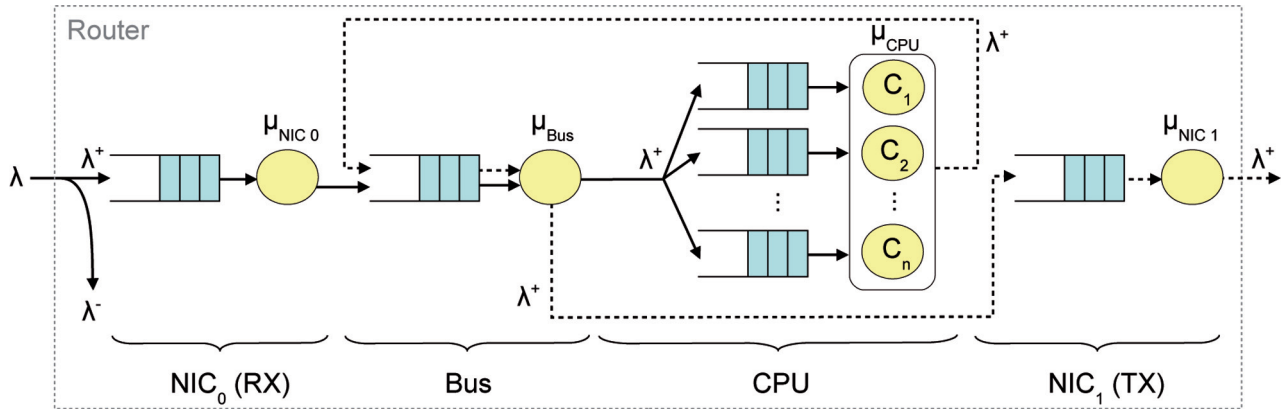
Fig. 2: Router Model with Packet Flow from NIC_0 to CPU (solid arrows) and from CPU to NIC_1 (dashed arrows).

sion for ns-3. We published further details regarding our modeling approach for resource management in resource-constrained nodes [27].

A Overview

Simulators are widely used for research and education. The reason is that deploying a testbed containing real networking devices and links is often expensive and time consuming. Researchers and designers can use simulators as a cost-effective approach to design, validate, and analyze their proposed protocols and algorithms in a controlled and reproducible manner [28].

Simulators can be classified into closed source and open source. Closed source simulators are often cost-intensive commercial products which need to be purchased. Open source simulators have the advantage that the source code is freely available and everyone can contribute to enhance it. In addition, open source simulators often reflect recent developments of new technologies in a faster way than commercial network simulators. There exist a variety of open source network simulators such as OMNeT++, ns-2, and ns-3 as well as closed source network simulators like OPNET [29].

Modern computers are multi-core or multi-processor systems and therefore parallel processing of protocol software becomes possible. Recent advances in computer architecture such as multi-core processors interconnected with high-speed links (e.g. Intel QPI) [26], integrated memory controllers, high bandwidth PCIe buses for the I/O transfer, and multiqueue multi-port NICs, allow high-speed parallel processing in network packet processors [3]. In multi-core systems, processes running simultaneously on different cores (or even threads running on the same core) may compete for shared resources (e.g., CPU, cache,

memory controller, and buses). This situation is called *resource contention* which can significantly degrade the performance in comparison to a contention-free environment. The effects of resource contention in multiprocessor and multi-core systems have been widely studied in the literature [30]–[32].

To the best of our knowledge, there is no support for modeling resource contention in network simulators though, evidently, resource contention must be modeled when realistic node behavior is required. Current node models of the existing network simulators typically assume unlimited resources and sequential packet processing. This limitation becomes problematic when resource-constrained nodes like software routers or sensor nodes are used and parallel processing of protocol software is an issue.

For instance, the node model of the network simulator ns-3 does not consider any node-internal resources (e.g. multi-core CPU) for the packet processing. Thus, intra-node latency and queueing behavior are not modeled. This has motivated us to elaborate a general concept for a detailed and thus realistic modeling of resource contention in network nodes.

Ns-3 is an open source discrete event simulator which is implemented in C++ for research and education in the networking area. It is rebuilt from scratch and is not an extension of ns-2. The main reasons for the popularity of ns-3 are its modularity, multi-technology support, and the simulation capabilities for large-scale scenarios. Ns-3 is capable of running simulation scenarios with more than 20,000 nodes, while ns-2 (version 2.33) is not able to simulate more than 8,000 nodes. Besides, ns-2 consumes more memory compared to ns-3 in a same simulation scenario [29]. Furthermore, in ns-3, packets can be saved to PCAP files, in a real packet format, making it wellsuited for real world integration. For the above reasons, we selected ns-3 for our studies.

B Theoretical Foundations

The packet processing of the system internal components like NICs, buses (QPI, PCIe) or CPU cores of the router can be modeled as a tandem queueing network as depicted in Fig. 2. Each system internal component possesses an incoming queue and a service rate μ. Following [2], [3], [7], [13] we assume that the CPU is the bottleneck. Therefore, Eq. (1) is essential where μ_C denotes the service rate of component C.

$$\mu_{CPU} = min\left\{\mu_{CPU}, \frac{\mu_{Bus}}{2}, \mu_{NIC\,0}, \mu_{NIC\,1}, \dots\right\} \quad (1)$$

The *offered load* is a specific sample of traffic which is applied to a device under test (DUT). Here, the offered load is characterized by the packet rate λ with a constant packet size. The packet rate splits into the accepted packet rate λ^+ and the dropped packet rate λ^- and therefore $\lambda = \lambda^+ + \lambda^-$. Due to our assumption that the CPU cores are the bottleneck, also $\lambda^+ = min\{\lambda, \mu_{CPU}\}$ holds.

If the router is not overloaded ($\lambda \leq \mu_{CPU}$) no packets are dropped ($\lambda^- = 0$). Otherwise, if the router is overloaded ($\lambda > \mu_{CPU}$) then packets get dropped ($\lambda^- > 0$). In this case, the accepted packet rate corresponds to the service rate of the CPU bottleneck ($\lambda^+ = \mu_{CPU}$). This means that we can derive the packet service time x of the bottleneck, here $x_{CPU} = \frac{1}{\mu_{CPU}} = \frac{1}{\lambda^+}$, based on real maximum throughput measurements in the testbed.

In this paper, we are interested in the maximum throughput of a software router. If we assume that the CPU cores are the bottleneck within the router, we can simplify our router model as depicted in Fig. 3. It consists of an incoming packet queue and multiple service units such as the CPU cores $C_1, \dots, C_n$. In our case, this simple model is applied to predict the maximum throughput. However, for instance, the packet latency in the router can also be analyzed by extending this model. Therefore, the packet delay of every node internal system like NIC and bus must be added. The packet delays at most of these components should be close to their service times because they are not overloaded and we assume that the CPU is the bottleneck. Evidently, the service time of the bottleneck is well approximated through the mean packet interdeparture time at the maximum throughput of the router.

The sojourn time T of a packet in the router is the time interval between the time t_{RX}, when the router receives a packet, and the time t_{TX}, when the router transmits this packet. Besides, the sojourn time consists of the waiting time w and the service time x.
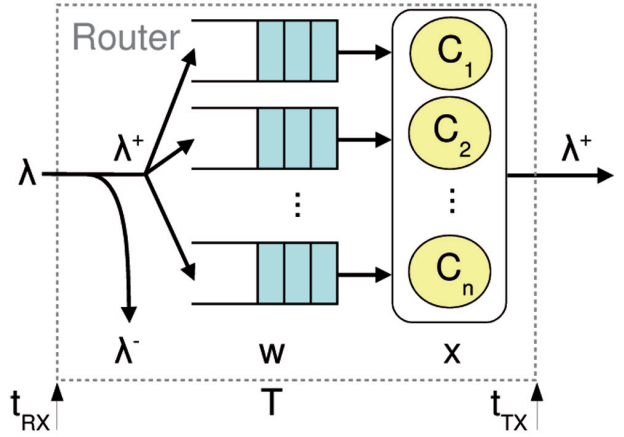

Fig. 3: Simplified Router Model with CPU Cores as Bottleneck.

$$T = t_{TX} - t_{RX} = w + x \quad (2)$$

The waiting time depends on the number of packets in the queue, their service times and the service strategy. Moreover, in many cases the service time x to process a packet is proportional to the frame size of the packet (e.g. packet copy). The frame size S (in Byte) refers to the packet size such that the frame size includes all headers (e.g. MAC, IP) without preamble, start of frame delimiter (SFD), and interframe gap (cf. Fig. 5). Therefore, we assume that x follows a linear function which is dependent on a constant part T_c and a frame size dependent part T_d.

$$x = T_c + S \cdot T_d \quad (3)$$

Furthermore, if we assume a stationary state and the offered load is larger than the maximum throughput of the router then the mean sojourn time $\overline{T}$ of a packet can be calculated based on Little's law [33]. The mean number of packets $\overline{N}$ in the router can be approximated based on the receive packet counter Z_{RX} and transmit packet counter Z_{TX} at a periodic sequence of observation times $t_i = i \cdot \Delta t$. If the router is overloaded and dropped load λ^- occurs then the accepted load λ^+ can be directly measured as the maximum throughput $\hat{D}_p$ in *packets per second* (pps) which brings us to the following equation.

$$\overline{T} = \frac{\overline{N}}{\lambda^+} \approx \frac{1}{j}\sum_{i=1}^{j}\frac{Z_{RX}(t_i) - Z_{TX}(t_i)}{\hat{D}_p} \quad , (\lambda^- > 0) \quad (4)$$

From our testbed measurements (cf. Fig. 8), we derive the heuristic relation that the maximum throughput $\hat{D}_b$ in *Gigabits per second* (Gbps) of our quad-core CPU router also follows a linear behavior. It is dependent on the number of used CPU cores k and the frame size S according to Eq. (5).

$$\hat{D}_b = (a \cdot k + a_0) \cdot S + (b \cdot k + b_0) \quad , (1 \le k \le n) \quad (5)$$

We assume that this heuristic also holds on arbitrary *n-core* CPUs if the offered load is uniformly split into k CBR flows with constant frame size S which are served by k CPU cores. The constant values for a, b, a_0 and b_0 are derived from our measurements of the real system, as it is done through the model calibration (cf. Section VI-B). The maximum throughput may also depend on other attributes (e.g. DMA transfer time, memory latency) which are omitted here to keep the model as simple as possible.

Based on this, the service time x per packet can be predicted. In addition, the Ethernet preamble (7 B), start of frame delimiter (1 B), and the interframe gap (12 B) must be considered in Eq. (6) (cf. Section V-C).

$$x = \frac{1}{\hat{D}_p} = \frac{(S + 7B + 1B + 12B) \cdot 8 \frac{Bit}{B}}{\hat{D}_b \cdot 10^9} \quad (6)$$

Furthermore, the actual service time x_{Core} per CPU core can be derived according to Eq. (7).

$$x_{Core} = k \cdot x = \frac{(S + 7B + 1B + 12B) \cdot 8 \frac{Bit}{B}}{((a + \frac{a_0}{k}) \cdot S + (b + \frac{b_0}{k})) \cdot 10^9} \quad (7)$$

This calculation of the per-packet service time is used in the case study simulations in Section VI which are based on our resource management model.

C Modeling of Intra-Node Resource Management

Our proposed resource management modeling approach is subdivided into three planes (Fig. 4).

- **Processing Plane**: At the lowest level there is the processing plane. It is composed of several task units *TU* which are connected with each other. Each task unit possesses specific processing functionalities *F* (e.g. decrease TTL) which require specific resources (e.g. CPU, memory, bus) and service times.
- **Resource Plane**: The resource plane consists of several resource pools (RP; e.g. Resource Pool CPU). Each resource pool contains resources *R* of the same resource type (e.g. CPU, memory or bus). Each resource pool is administered by exactly one local resource manager.
- **Resource Management Plane**: Several local resource managers (LRM; e.g. Local Resource Manager CPU) are located in the resource management plane. Above

all, exactly one global resource manager (GRM) exists to coordinate the local resource managers if a task unit requests several shared resources.
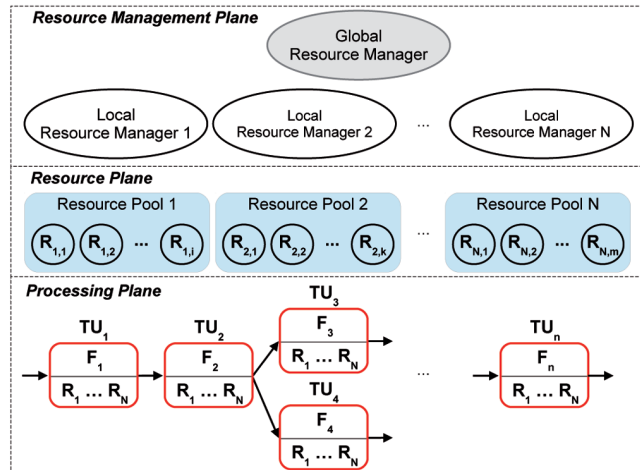

Fig. 4: Resource Model Planes.

1) Task Unit: A Task Unit (TU) is an entity which encapsulates functionality (e.g. IP processing) with uniform resource requirements. Incoming packets are waiting in the incoming queue Q_{in} of a task unit for being processed. At least one resource is needed to execute the functionality corresponding to the task unit for this packet. If currently not all of the required resources are available, the packet waits until the required resource(s) become(s) available. The service time of the task unit may depend on the packet-processing workload which can be characterized by the packet size and the type of packet-processing (e.g. IP routing, IPsec encryption). After processing the packet, it is enqueued in the task unit's outgoing queue Q_{out} to be processed by the next task unit.

A task unit can be subdivided into several task units to model specific effects in more detail (e.g. bus contention). This makes our resource management model flexible and extensible. We advise to use as few task units as possible to obtain simple models for efficient simulations.

2) Resource Manager: A Resource Manager (RM) is an entity which coordinates between multiple task units based on their task unit priority. We distinguish between three levels of detail in resource management modeling:

- **No Resource Manager**: The task unit(s) possess(es) dedicated resources (e.g. NIC uses its own processor). In this case, neither the global resource manager nor any local resource manager is required.
- **Local Resource Manager (LRM)**: A local resource manager is required if at least two task units share the same resource.

– **Global Resource Manager (GRM)**: A task unit requests the global resource manager if several shared resources of different resource types are needed.

3) Interactions between Resource Manager and Task Unit:

The interactions between the resource manager and the task unit(s) are based on different resource management messages.

– **ResourceRequest (REQ):** The task unit sends this message to its resource manager to apply for resources.

– **ResourceReply (REP):** This message is sent from the resource manager to the task unit in response to a REQ to allocate a specific resource (e.g. CPU) to the task unit.

– **ResourceRelease (REL):** The task unit sends this message to the resource manager to give back an allocated resource.

– **ResourceRevoke (REV):** This message is sent from the resource manager to the task unit to withdraw a resource which is currently occupied by this task unit.

Further details of our modeling approach of resource management are published in [27].

V Performance Evaluation with Measurements

In this section we focus on the performance evaluation with measurements on real systems. First, we set the requirements for our measurement setup. Second, we present our measurement tools used to generate and measure network traffic.

A Measurement Setup

The methodology for measuring the performance of a networking device is described in RFC 2544 [14]. It gives guidelines for measuring standalone networking devices from a black box point of view. The measured device is referred to as device under test (DUT). The document covers various performance indicators including throughput, latency, packet bursts, and system recovery time. As described in Section IV-B, the maximum throughput of a router is the fastest packet processing rate at which there are still no dropped packets and thus the number of packets entering and leaving the DUT are equal. Therefore, we need to carry out multiple measurements with different frame rates to achieve the maximum throughput. RFC 2544 also specifies

that Ethernet measurements need to be performed at varying frame sizes of at least 64, 128, 256, 512, 1024, 1280, and 1518 B. For our test case, this means performing multiple measurements at diverse packet rates and frame sizes and counting of the packets entering and leaving the device under test.

Networking devices are very complex, so there are plenty of side effects that can influence our measurements. First, we have to make sure the router knows all information needed for packet forwarding. Therefore, we populate the router's static routing table and ARP table before starting a measurement. Second, we avoid cross-traffic, e.g. by using statically configured interfaces thus avoiding DHCP messages. Our network interface cards perform several techniques that influence incoming and outgoing packets (cf. Section III-C). Third, we disabled Ethernet flow control on all devices, which influences the transmission of data, especially in overload situations. Finally, we want to make sure that the router, that is the DUT, behaves in a deterministic way, so we disabled advanced CPU features on the router machine. In particular, we disabled *Turbo Boost*, which influences the CPU clock speed but would disrupt the results. We even measured a negative influence on μ_{CPU} for evenly distributed load. We also deactivated *Hyper-Threading*, which has no benefit in our case, as already one thread is able to completely utilize the core.

B Traffic Generators

Having defined the requirements for our throughput measurements, we need to find test tools that meet these requirements. We need to generate 64 B to 1518 B frames at a constant rate, that should scale up to the link speed. On our 10 Gbps hardware, this translates to more than 14 Mpps at a size of 64 B. Traffic generators like UniLoG [34] focus on building manifold traffic, i.e. in case of destination and source IP addresses, payload, or temporal or size distribution. Other load generators focus on producing very high numbers of packets. However, even commonly used traffic generators like pktgen or Iperf produce a limited amount of packets which did not saturate our links. Due to the overhead produced by the OS network stack they are able to produce about 1 Mpps. A load generator based on Click, which does not rely upon the Linux network stack allows traffic generation at a rate of about 4 Mpps. The *pfsend* packet generator from the PF_RING DNA software repository [6], uses a zero-copy technique and is capable of filling our links using 64 B frames. This *pfsend* packet generator was used for all measurement experiments presented in this paper to produce packets at a constant bit rate (CBR).

C Measurement

In order to perform the throughput measurement, we need to count packets entering and leaving the router. Therefore, we need a measurement tool capable of counting packets at a rate of more than 14 Mpps. Complex traffic analysis tools that allow for a detailed traffic analysis and offer manifold traffic statistics have problems dealing with these rates. If the analysis tool cannot cope with the offered load, packets get dropped on the measurement host, which directly influences our results. None of the tested traffic analysis tools – not even those using a zero copy technique – could handle that many packets. Our network interface cards keep traffic statistics, e.g. the number of received and transmitted packets, in hardware. The interface driver makes these statistics available via a Linux pseudo file system. Packet counters can be obtained by accessing these information[1]. The number of packets at the sink is obtained by adding the value of packets dropped by the NIC with the value of successfully received packets. The packet rate can be calculated easily from periodical updates of the hardware statistics.

The throughput D_b can be calculated based on the packet rate λ and the frame size S. Besides, the Ethernet preamble (7 B), the start of frame delimiter (SFD; 1 B) and the interframe gap (12 B) must be added to the frame size for each packet.
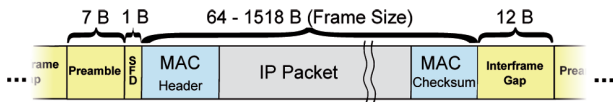

Fig. 5: Ethernet Frame Structure.

$$D_b = (S + 7\,B + 1\,B + 12\,B) \cdot 8\frac{Bit}{B} \cdot \lambda \qquad (8)$$

Eq. (8) results in a maximum throughput of 14.88 Mpps that can be theoretically achieved with a 10 Gbps link and 64 B sized frames. We illustrated the Ethernet frame structure in Fig. 5. Neglecting the size of the preamble and the interframe gap is an error that can distort results. For instance, RouteBricks [2] claimed a packet rate of 18.96 Mpps at 9.7 Gbps caused by this calculation error.

---

1 i.e. rx_missed_errors, rx_packets, and tx_packets, found in /sys/class/net/dev_name/statistics.

VI Case Study

In this section we evaluate the packet processing performance of a modern quad-core software router. On the one hand, we conduct real measurements in a testbed. On the other hand, we use simulations based on our ns-3 *resource management* extension [27]. The case study aims for the calibration, validation, and verification of our ns-3 extension based on real testbed measurements.
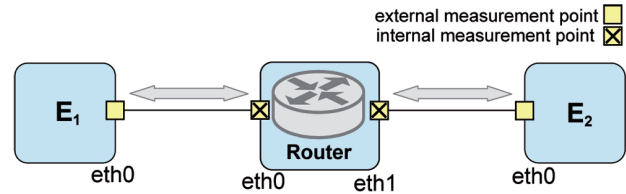

Fig. 6: Case Study Scenario with a Resource-Constrained Software Router.

A Scenario

The testbed and simulation scenario consists of two end devices E_1 and E_2 acting both as a combination of a *Load Generator* and a *Sink* and a software *Router* in between serving as device under test. The end devices are connected via dedicated 10 Gbps Ethernet links to the router (Fig. 6).

We apply bidirectional CBR traffic of 1–4, and 100 flows with constant frame sizes of 64, 128, 256, 512, 1024, and 1518 B. The frame size is denoted as the Ethernet frame size without preamble (7 B), start of frame delimiter (1 B), and interframe gap (12 B). The odd flows (e.g. first flow) are sent from the load generator on E_1 to the sink on E_2 whereas the even flows (e.g. second flow) are transmitted from the load generator on E_2 to the sink on E_1. Thus, we are able to exploit the full-duplex capability of the 10 Gbps links which enables us to apply a maximum offered load of 20 Gbps. The offered load is greater than the maximal throughput $\hat{D}_{Router}$ of the software router under test and is uniformly distributed among the number of flows. As a result, the maximum throughput achieved is depicted in dependence on the load offered to the software router.

1) Testbed Measurements: We implemented the software router using commodity server hardware. The system has been equipped with 16 GiB RAM and one Intel Xeon E3-1230 V2 CPU operating at a clock speed of 3.3 GHz. The schematic structure of the CPU can be seen in Fig. 1. The Xeon E31230 V2 CPU is based on the Intel Ivy Bridge

architecture and comes with four cores. Via an 8-lane PCIe 2.0 interface we attached a dual-port Intel 10 Gbps X520-SR2 network interface card (NIC). This high-end NIC comes with many features and offloading techniques. In our measurements we only make use of the Receive Side Scaling (RSS) feature.

Based on this hardware setup, our software router is implemented using Linux with IP forwarding enabled. We use the GRML Linux distribution along with the 3.7 Linux kernel. The measurement of the Linux IP forwarding performance was selected due to its high relevance in practice. Aside from that we used version 3.14.5 of the *ixgbe* driver (released in April 2013), since we discovered its performance is significantly better than previous and later versions.

We use the *pfsend* load generator to produce artificial CBR traffic at packet rates that scale up to the link speed of 10 Gbps. As explained above, the produced traffic consists of 1-4 and 100 flows with evenly distributed packet rates. These flows are crafted in a way that they are distributed to distinct cores by the RSS algorithm. In case of 1-4 flows, this effectively means that each flow is processed in a dedicated core. For the 100 flow test scenario, the flows were not specifically crafted to be distributed equally among the four cores of our test system, instead we let the RSS algorithm distribute 100 randomly chosen flows. Packet counters are implemented on both end devices, depicted as the external measuring points in Fig. 6. The results of the throughput tests are displayed in Fig. 8.

2) Simulation Measurements: Our ns-3 resource management extension is applied to the router under test. The corresponding router model is derived from our modeling approach (cf. IV-C). It is illustrated in Fig. 7. The limited resources of the modeled router are represented in the resource plane. The resource pool RP_{Core} of the CPU cores contains four CPU cores $C_1, \ldots, C_4$ which are administered by the resource manager RM_{Core} in the resource management plane. The packet processing in a software router (e.g. IP table lookup, firewalling) is modeled in the processing plane which consists of the task units $TU_{Distributor}$, $TU_{Core\,1}, \ldots, TU_{Core\,4}$ as well as the incoming queues $Q_1, \ldots, Q_4$ in between.

To process a packet, a task unit needs specific resources. Therefore, the task unit has to apply to the resource manager for allocating it. For instance, $TU_{Core\,1}$ requires a CPU core from RP_{Core} where (in this simplified case) for each of $TU_{Core\,i}$ $(1 \leq i \leq 4)$ a CPU core is available. Thus, RM_{Core} allocates a core resource, e.g. C_1, to $TU_{Core\,1}$. After that, $TU_{Core\,1}$ starts to process the incoming packets

from Q_1. Resource contention for shared resources may occur if we would take into account a smaller number of CPU core resources or when an additional task unit also requires a CPU core. However, resource contention among the packets occurs with respect to the corresponding task unit which limits the maximum throughput. The task unit functionality *Processing* consumes simulation time corresponding to the required service time of the current packet (cf. Eq. (7) in Section IV-B).
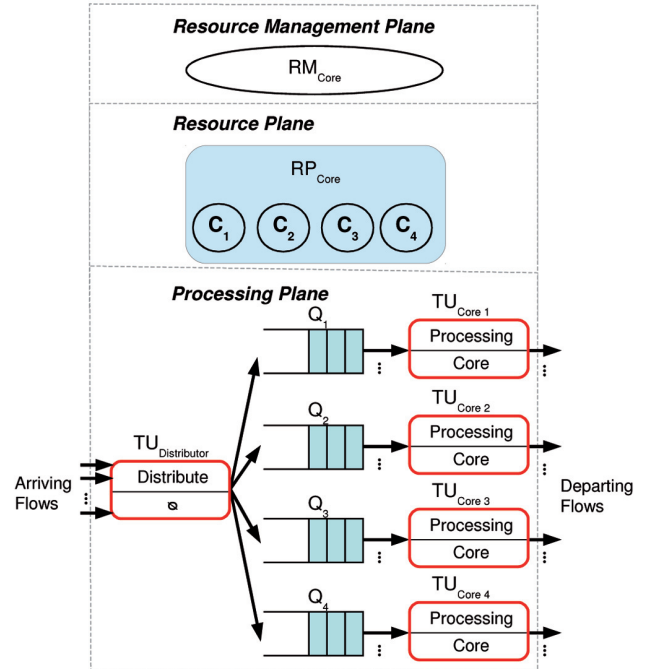

Fig. 7: Resource-Constrained Router Model.

Besides, 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. Based on that, the $TU_{Distributor}$ maps each flow to a specific incoming queue Q_i of a task unit $TU_{Core\,i}$ $(1 \leq i \leq 4)$. In consequence, every packet of a specific flow is served by the same task unit corresponding to a specific CPU core.

In this case study, we assume that the CPU cores are the bottleneck as it was observed in a lot of earlier measurements by us [4] and other researchers [2]. Hence, the functionality *Distribute* of task unit $TU_{Distributor}$ (which is usually done by the NIC controller and DMA) adds no additional service time and needs no shared resources (denoted by $\oslash$ in Fig. 7). Therefore, only one resource type (namely the CPU cores) was considered. However, other resource types and intra-node effects (e.g. NIC Tx/Rx queues, cache misses) can be modeled to set up more complex case studies. Furthermore, the load generator and sink have no resource constraints, but the DUT pos-

sesses a limited number of CPU cores. The service time of a packet in the router depends on its size and the number of used CPU cores in the router (cf. Eq. (7) in Section IV-B). The service time parametrization of the router to process a packet is derived from real testbed measurements as described in Section VI-B.

B Calibration of Software Router Simulation Model

Model calibration is the process of setting the well-defined parameters of the simulation model with respect to a specific real system. The determination of the model parameters is based on measurement results of the modeled system.

However, there are measurement points which are not applicable for model calibration. For instance, when applying data transmissions at a high level of offered load between the load generator and the sink then the 10 Gbps Ethernet link becomes bottleneck instead of the CPU cores.

In this case study, we only need four measurement points for the calibration of the router simulation model. These calibration points are depicted as encircled points in Fig. 8(b) as well as listed in Table I(a). All other results thereafter can be predicted with the help of simulations based on the calibrated router model.

Table 1: Measured maximum throughput ($\hat{D}_{meas}$) for varying number of flows (F) and frame sizes (S) used for calibration of the model parameters (a, a_0, b, b_0)

F	S [B]	$\hat{D}_{meas}[Gbps]$
1	64	1.1928
1	512	7.5099
4	64	4.0612
4	256	13.3304

(a) Measurement Points

a	0.011392
a_0	0.002708
b	0.227037
b_0	0.063321

(b) Router Model Parameters

Based on these measurement values and according to our service time calculation of the router model (cf. Eq. (7) in Section IV-B) the calibration parameters a, b, a_0, and b_0 of the router model are derived which are depicted in Table I(b).

C Validation of Software Router Simulation Model

As mentioned in Section VI-B, our introduced router model has been calibrated by means of real testbed measurements of a router based on a modern quad-core processor. In this section we now want to investigate whether our maximum throughput predictions do really represent sufficiently valid predictions of the real system behavior. For this purpose we want to compare our throughput predictions (given in Gbps and Mpps) with the measured values of throughput for different frame sizes.

Fig. 8(a) and 8(b) illustrate the maximum throughput predicted by our simulation model for a quad-core processor system and, for comparison purpose, the throughput values actually measured in our testbed. The x-axis shows the frame size in Byte. The y-axis represents the measured and simulated maximum throughput of the router in Mpps and in Gbps respectively. The chart shows that the maximum throughput is not significantly dependent on the frame size because the routing table lookup overhead is similar for small and large packets. However, the maximum throughput of a multi-core software router strongly depends on the number of used cores and their distribution to the multiple cores to benefit from parallel processing capabilities.

We can observe that the simulation results coincide with the measured values. In Table II we show the measured maximum throughput $\hat{D}_{meas}$, the simulated maximum throughput $\hat{D}_{sim}$, and the relative error *Err* in percentage where $Err = \frac{\hat{D}_{sim} - \hat{D}_{meas}}{\hat{D}_{meas}}$. The confidence bounds were omitted because the simulation results are based on CBR traffic (cf. Section VI-A) and do not show large variance. The mean deviation is ca. 0.95% which indicates that our ns-3 extension is precise enough to produce realistic simulation results which is part of a successful model validation process. The bold values in Table II denote the used calibration points, whereas values are omitted where the 10 Gbps link is the bottleneck.

Table 2: Relative error (*Err*) of measured ($\hat{D}_{meas}$) and simulated ($\hat{D}_{sim}$) maximum throughput for varying number of flows (*F*) and frame sizes (*S*)

F	S [B]	$\hat{D}_{meas}$[Gbps]	$\hat{D}_{sim}$[Gbps]	Err [%]
1	**64**	**1.1928**	**1.1943**	**0.12**
1	128	2.0936	2.0968	0.15
1	256	3.9097	3.9033	−0.16
1	**512**	**7.5099**	**7.5237**	**0.18**
1	1024	9.9379	9.9276	−0.10
1	1518	9.9682	9.8951	−0.73
2	64	2.2265	2.1516	−3.36
2	128	3.9251	3.7849	−3.57
2	256	7.3182	7.0583	−3.55
2	512	14.0514	13.5834	−3.33
3	64	3.1858	3.1073	−2.47
3	128	5.6076	5.4744	−2.37
3	256	10.4578	10.1926	−2.54
3	512	16.7076	16.4304	−1.66
4	**64**	**4.0612**	**4.0686**	**0.18**
4	128	7.1429	7.1573	0.20
4	**256**	**13.3304**	**13.3450**	**0.11**
100	64	4.0191	4.0686	1.23
100	128	7.0684	7.1574	1.26
100	256	13.1533	13.3461	1.47

Although, we assumed a heuristic relation in the calibration of our resource management model for a quad-core router, (cf. Section IV-B), the validation experiments described here demonstrate that already simulation models which have been elaborated without much expenditure lead to realistic performance predictions if an adequate modeling of resource contention is carried out. The realistic calibration and parameterization of the resource management model however is highly important in the current scenario of this case study in order to be able to achieve a satisfying level of model validity.

D Prediction Based on Software Router Simulation Model

By applying our calibrated and validated router model, it is possible to forecast the maximum throughput performance of software routers with faster CPUs. As the trend to larger number of CPU cores can be expected to continue

instead of significantly higher CPU clock frequencies, we assume that the CPU cores remain the bottleneck. This implies that intra-node systems like buses and caches challenge the growing number of cores.

We keep the simulation scenario as used in Section VI-A but now the software router is based on a 8-core processor architecture which has to process two 50 Gbps links. A link speed of 50 Gbps may be supported by future NICs, but can already be implemented today using multiple 10 Gbps interfaces.

Fig. 9(a) and 9(b) illustrate the predicted maximum throughput by our simulation model dependent on the frame size for different number of flows. The x-axis shows the frame size in Byte. The y-axis represents the simulated maximum throughput of the router in Mpps, and in Gbps respectively. Among others, this forecast shows that such a software router may create a bottleneck if less than six flows are processed. Besides, our results also show that small frame sizes strongly limit the maximum throughput of the software router, even in cases when all cores are used. However, with a realistic mix of frame sizes and number of flows the results indicate that the software router achieves a reasonably high maximum throughput.

VII Summary and Future Work

In this paper, we measured and simulated the performance of software routers based on current multi-core architectures. For identifying bottlenecks or to predict the performance of such systems, the node models for resource-constrained nodes (e.g. software routers, sensor nodes, smartphones) currently used in simulators such as ns-3 are by far too simplistic. Therefore, we introduced a new approach for modeling the resource contention in resource-constrained nodes at different levels of detail. Based on that, we successfully extended ns-3 for intra-node resource management. We calibrated and validated this model in a case study. We measured the software router performance on off-the-shelf multi-core hardware for comparison. We also described the challenges we had to address when performing measurements at high packet rates and our solutions to these problems. The case study showed that we are able to predict performance behavior of the tested software router in a realistic manner even in the case when parallel processing with multi-core processors is applied. Our comparisons with real system measurements substantiate our claim of being able now to observe realistic model behavior. We used the model to predict how a growing number of CPU cores will affect the ability of
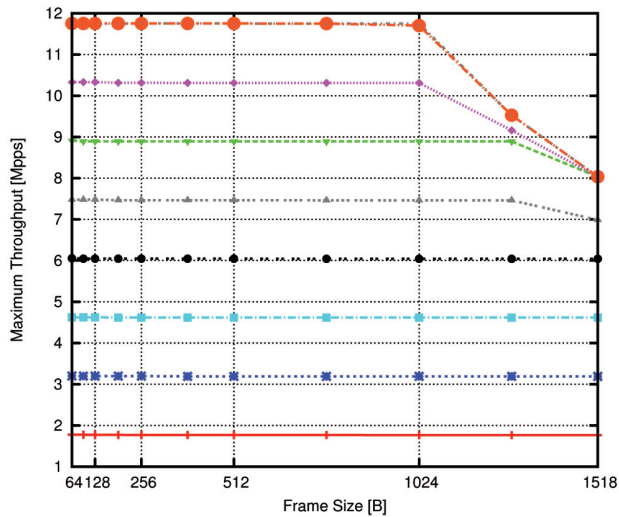
Validated Model-Based Performance Prediction of Multi-Core Software Routers ── **105**
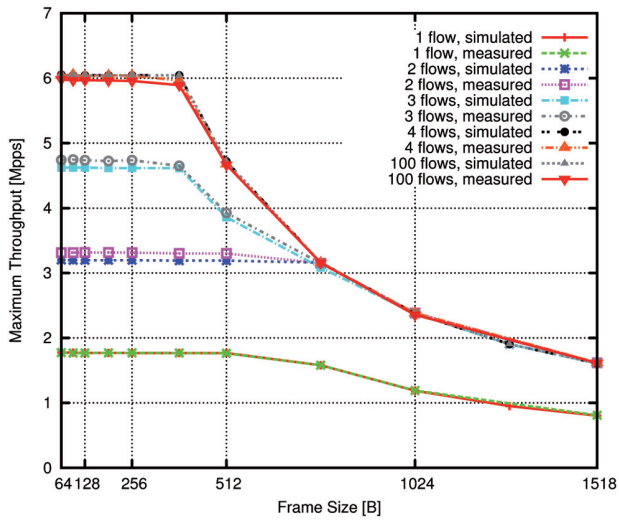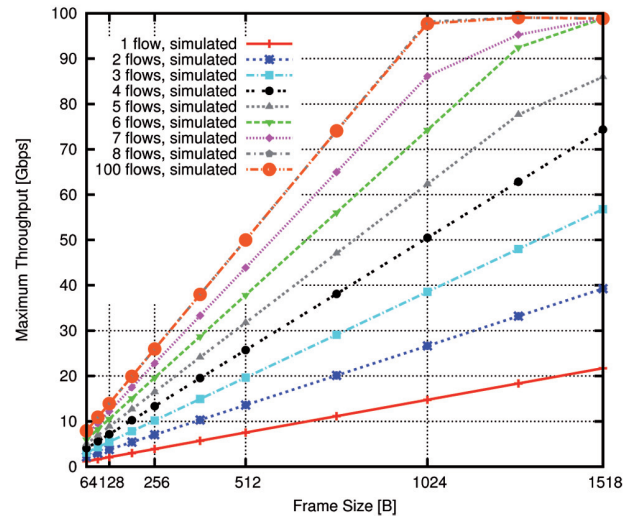

(a) Maximum Throughput in Mpps
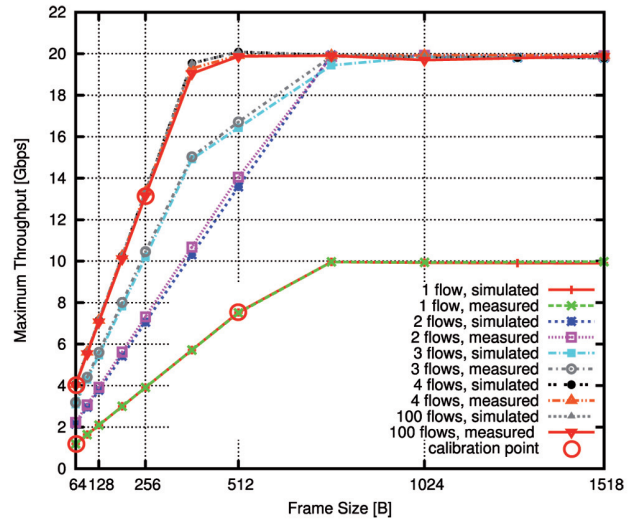
(b) Maximum Throughput in Gbps

Fig. 8: Simulation Results for the Maximum Throughput of the Modeled Resource-Constrained Software Router in Comparison to Real Testbed Measurements.

(a) Maximum Throughput in Mpps (Legend: cf. 9(b))

(b) Maximum Throughput in Gbps

Fig. 9: Simulation Prediction of the Maximum Throughput of a Modeled Resource-Constrained Software Router with 8 Cores and 50Gbps Network Links.

software routers to deal with higher loads – regardless if it is due to the growing speed of single network links or a growing number of NICs within a software router. Our results also revealed that in certain scenarios software routers have free resources, which could be used for more advanced packet processing, such as encryption.

Our plans for the future comprise to refine our resource-constrained software router model in terms of the relevant details. Therefore, we will need to carry out more fine-grained measurements, modeling, and simulation. The measurement and simulation of the packet sojourn time will be one of the next steps to analyze the latency behavior of a software router. In addition to our existing black-box measurements, we want to investigate the routing software using code inspection and profiling. We intend to identify the 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

References

1   C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking," in *1st Workshop on Hot Topics in Software Defined Networks (HotSDN)*, August 2012.

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   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.

4   T. Meyer, F. Wohlfart, D. Raumer, B. E. Wolfinger, and G. Carle, "Measurement and Simulation of High-Performance Packet Processing in Software Routers," in *Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und verteilten Systemen, 7. GI/ITG-Workshop MMBnet*, September 2013.

5   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.

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

7   L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," in *USENIX Annual Technical Conference*, April 2012.

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

9   A. Bobrek, J. Pieper, J. Nelson, J. Paul, and D. Thomas, "Modeling Shared Resource Contention Using a Hybrid Simulation/Analytical Approach," in *Design, Automation and Test in Europe Conference*, vol. 2, February 2004, pp. 1144–1149.

10   O. Sokolsky, "Resource Modeling for Embedded Systems Design," in *2nd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, May 2004, pp. 99–103.

11   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.

12   M. Bjorkman and P. Gunningberg, "Performance Modeling of Multiprocessor Implementations of Protocols," *IEEE/ACM Transactions on Networking*, vol. 6, no. 3, pp. 262–273, June 1998.

13   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.

14   S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544 (Informational), Internet Engineering Task Force, March 1999.

15   M. Handley, O. Hodson, and E. Kohler, "XORP: An Open Platform for Network Research," in *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, January 2003.

16   "Quagga Routing Suite," http://www.nongnu.org/quagga/, November 2013.

17   "BIRD Internet Routing Daemon," http://bird.network.cz/, November 2013.

18   "Vyatta," http://www.vyatta.org/, November 2013.

19   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.

20   G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "ServerSwitch: A Programmable and High Performance Platform for Data Center Networks," in *8th USENIX Conference on Networked Systems Design and Implementation*, April 2011.

21   "Open vSwitch," http://openvswitch.org/, November 2013.

22   H. Khosravi and T. Anderson, "Requirements for Separation of IP Control and Forwarding," RFC 3654 (Informational), Internet Engineering Task Force, November 2003.

23   Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow Switch Element for Click," in *Symposium on Click Modular Router*, November 2009.

24   E. Kohler, "Click for Measurement," UCLA Computer Science Department, Tech. Rep., February 2006.

25   G. Varghese, T. Piazza, and H. Jiang, "Intel Next Generation Microarchitecture Codename Ivy Bridge," Intel Corporation, Tech. Rep., September 2011.

26   B. Mutnury, F. Paglia, J. Mobley, G. Singh, and R. Bellomio, "Quick Path Interconnect (QPI) Design and Analysis in High Speed Servers," in *IEEE Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, October 2010.

27   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)*, July 2013.

28   J. Pan and R. Jian, "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   C. Xu, X. Chen, R. P. Dick, and Z. M. Mao, "Cache Contention and Application Performance Prediction for Multi-core Systems," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2010, pp. 76–86.

31   S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," in *15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, New York, USA, March 2010, pp. 129–142.

32   R. Hood, H. Jin, P. Mehrotra, J. Chang, J. Djomehri, S. Gavali, D. Jespersen, K. Taylor, and R. Biswas, "Performance Impact of Resource Contention in Multicore Systems," in *IEEE International*

Symposium on Parallel Distributed Processing (IPDPS), April 2010, pp. 1–12.

33　J. D. Little, "A Proof for the Queuing Formula: L= λW," *Operations Research*, vol. 9, no. 3, pp. 383–387, May 1961.

34　A. Kolesnikov, "UniLoG: A Unified Load Generation Tool," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, ser. Lecture Notes in Computer Science, March 2012, vol. 7201, pp. 253–257.

Daniel Raumer: Technische Universität München, Department of Computer Science, Network Architectures and Services

Torsten Meyer: Universität Hamburg, Department of Computer Science, Telecommunications and Computer Networks

Bernd E. Wolfinger: Universität Hamburg, Department of Computer Science, Telecommunications and Computer Networks

Florian Wohlfart: Technische Universität München, Department of Computer Science, Network Architectures and Services

Georg Carle: Technische Universität München, Department of Computer Science, Network Architectures and Services