Measurement and Simulation of High-Performance Packet Processing in Software Routers

Torsten Meyer[1], Florian Wohlfart[2], Daniel Raumer[2], Bernd E. Wolfinger[1] and Georg Carle[2]

[1]Universität Hamburg, Department of Computer Science, Telecommunications and Computer Networks
{meyer|wolfinger}@informatik.uni-hamburg.de
[2]Technische Universität München, Department of Computer Science, Network Architectures and Services
{wohlfart|raumer|carle}@net.in.tum.de

***Abstract*—The possibility of using flexible and cost-efficient commodity hardware instead of expensive custom hardware has generated wide interest in software routers. Performance measurement and simulation have become 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 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**

I. Introduction

The performance of commodity PC hardware has increased rapidly. Off-the-shelf systems can be used 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. Thereby commodity hardware is cheaper than specialized server 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. In network related developments this shows at examples like the IETF NETCONF WG or the Open Networking Foundation, where new concepts (in this case protocols that enable the configuration of network devices) were rapidly implemented and tested in software, while hardware implementations were not available.

Nevertheless the benefits of software routers comes with the drawback of smaller number of ports and lower throughput rates. RouteFlow [1] combines 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]. For the use of computer systems in high-speed networks the traffic of 1 Gbps and 10 Gbps 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 operating system (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 usually there is a lack of a model 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 applicable for manifold simulation scenarios and be scalable 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 widely used network simulator ns-3. After calibrating and validating our model based on our real testbed measurements, we evaluate and predict the performance of current and future software router architectures.

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 the paper and give an outlook in Section VII.

II. Related Work

Scientists aiming for low level optimizations [3]–[5] 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 [6]–[10] can help to understand the related effects and performance factors.

Some projects consider surveying software router implementations as task with the goal of providing hints for future optimizations [2], [11], [12]. In context of the RouteBricks 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 [3] utilizes the GPU to speed up packet processing. PF_RING [4] and netmap [5] 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. [6] 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. [7] used a hybrid simulation/analytical approach for modeling shared resource contention in Programmable Heterogeneous Multiprocessor (PHM) systems. Besides, Sokolsky [8] 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. [9] proposed a high-level approach for modeling an observed system behavior with little knowledge about the system internal structure or operation. This is done by adequately selecting the parameters of a set of queueing systems and queueing networks. Bjorkman and Gunningberg [10] investigated the effects of locks and memory contention which are major performance bottlenecks in multi-processor systems. They also 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 [13]. Bolla and Bruschi [12] 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. [11] 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 [14], Quagga [15] and BIRD [16] are the most well-known ORs. The Vyatta Open-Firmware-Router [17] 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 [18] was used mainly for scientific research. Other examples for PC-based packet processing are ServerSwitch [19], as proof of concept for flexible packet switching in cloud data centers, and Open vSwitch [20] 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 and fault-tolerant 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 to track the networks reachable via its neighbors in a routing table. In summary the duties of an IP router are twofold: first, it needs to learn its routing table (either via static rules or a distributed 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* [21].

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 not time-critical, the control plane is usually implemented in software, that is running on general-purpose processors, even in dedicated hardware routers [21].

On the other hand, the tasks of the forwarding plane are rather simple but critical in terms of packet 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. This makes the term forwarding plane misleading. However, these tasks are simple enough to be implemented using special-purpose chips (ASICs) in hardware routers [21]. 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 the related work section there are several mature software router implementations for UNIX-like platforms, such as Linux, FreeBSD, NetBSD, and Solaris. These platforms come with full forwarding plane functionality. They natively support IP forwarding according to the system routing table. 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 [15], BIRD [16], and the eXtensible Open Router Platform (XORP) [14] 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 implementation of the OS. The forwarding plane implementation of XORP relies on Click, which we will discuss in the upcoming section.

B. Software-Based Routing Performance Optimizations

The forwarding plane functionality in UNIX-like systems 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 [18] provides a replacement for the OS network stack with its own forwarding plane implementation. 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. Click 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. This means Click is very flexible and allows to build almost any kind of packet processing software – such as an IP router.

Building 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 [18]. 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 [5] or an OpenFlow switch element [22]. Click elements such as a load generator and a load sink contributed by the author of [23] 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 lead to the New API (NAPI). The new software techniques 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 still 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 copying 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. Zero-copy packet processing aims to avoid costly copy operations between DMA accessible buffers, kernel-level buffers, and user space buffers. Prominent examples are PF_RING [4] coming from Linux and netmap [5] 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. Components were integrated with others for sake of communication optimization and the increase of density. The memory controller is placed on-chip 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).

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 Segmentation 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. 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

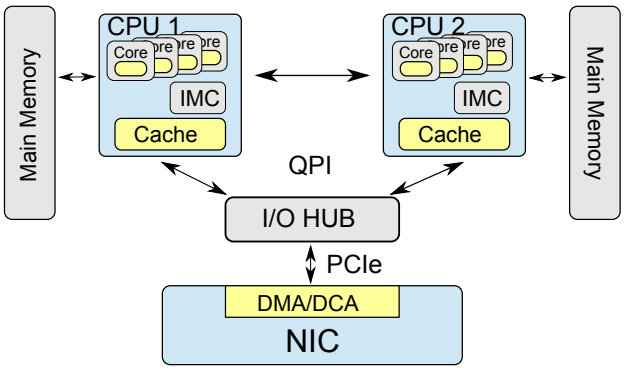


Fig. 1. Intel Hardware Architecture

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.

IV. Performance Evaluation with Simulations

In this section, we give an overview of current network simulators with respect to intra-node resource contention modeling. 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 extension for ns-3. Further details regarding our modeling approach for resource management in resource-constrained nodes were published by us [24].

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 [25].

Simulators can be classified into closed source and open source. Closed source simulators are often cost-intensive commercial products which need to be licensed. 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 [26].

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) [27], integrated memory controllers, high bandwidth PCIe buses for the I/O transfer, and multi-queue multi-port NICs, allow high-speed parallel processing in network packet processors [11]. 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*. Resource contention can significantly degrade the performance in comparison to a contention-free environment. The effects of resource contention in multi-processor and multi-core systems have been widely studied in the literature [28]–[30].

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 network simulator ns-3 only offers a very simplified model to take into account the intra-node resources available in a network node. 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 [26]. Furthermore, in ns-3, packets can be saved to PCAP files, in a real packet format, making it well-suited for real world integration. For the above reasons, we select ns-3 for our studies.

B. Theoretical Foundations

According to Fig. 1, 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.

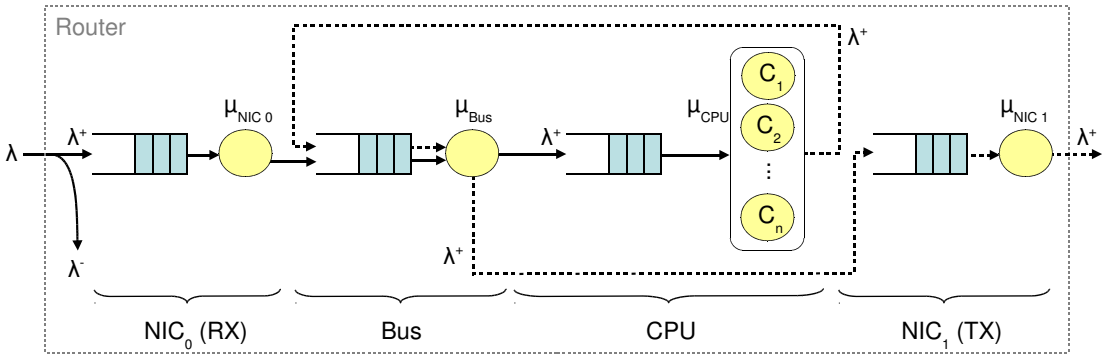


Fig. 2. Router Model with Packet Flow from NIC_0 to CPU (solid arrows) and from CPU to NIC_1 (dashed arrows)

Each system internal component possesses an incoming queue and a service rate μ. According to RouteBricks [2], we assume that the CPU is the bottleneck. Therefore, the following equation is essential where μ_C denotes the service rate of component C.

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

We assume that 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 ($\lambda \leq \mu_{CPU}$) then no packet must be dropped ($\lambda^- = 0$). Otherwise, if the router is overloaded ($\lambda > \mu_{CPU}$) then packets must be 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 $C_1 \cdots C_n$.

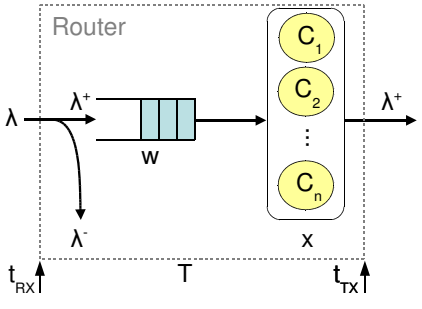


Fig. 3. Simplified Router Model with CPU Cores as Bottleneck

In our case, this simple model is sufficient to predict the maximum throughput. Nevertheless, it is just a throughput model and not a delay model. Therefore, the packet sojourn time in the router cannot be analyzed with this model. For analyzing the total packet sojourn time, 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 inter-departure 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.

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

The waiting time depends on the number of packets in the queue, their service times and the service strategy. Moreover, the service time x to process a packet typically follows a linear behavior in terms of the packet size. Therefore, let us assume in the following that x depends on a constant part T_c and a packet 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 [31]. 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.

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

Based on our testbed measurements (cf. Fig. 10), 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 packet size S according to Eq. (5) because packets belonging to the same flow are always mapped by RSS to the same CPU core.

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

We assume that this heuristic exists on a n-core CPU if the offered load is uniformly split into k CBR flows with constant packet 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). Besides, 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. From these values the expected service time x per packet can be predicted. Besides, the Ethernet preamble, start of frame delimiter and the interframe gap must be considered (cf. Section V-C).

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

This per-packet service time calculation 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 model is subdivided into three planes (Fig. 4).

- **Processing Plane**: At the lowest level there is the processing plane which 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 time.
- **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.

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

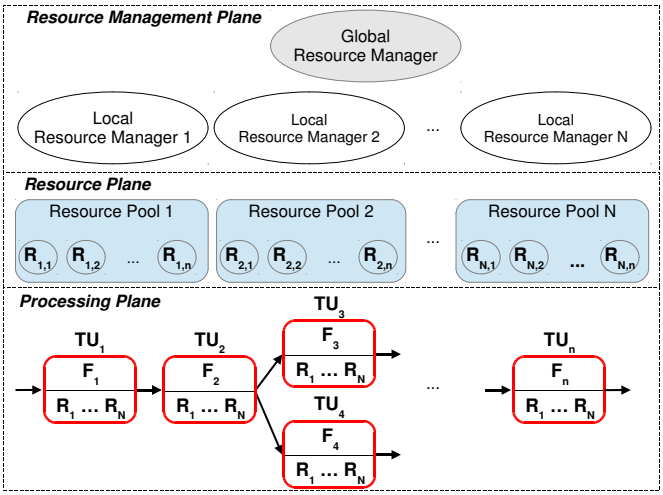


Fig. 4. Resource Model Planes

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. However, there should be 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)**: If at least two task units share the same resource, a local resource manager is required.
- **Global Resource Manager (GRM)**: If several shared resources of different resource types are needed, the task unit requests the global resource manager.

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.

D. Implementation of Intra-Node Resource Management

We implemented our resource management concept proposed in Section IV-C as a *Resource Management* extension for the network simulator ns-3. As ns-3 is modularly organized, the implementation of this extension is not restricted to any specific scenario. We provide a general framework for modeling intra-node resources and their contention. This extension can be used for modeling arbitrary scenarios related to intra-node resources like CPU cores, memory controllers, and internal buses.

Resources are modeled as typed entities. For example, in a given simulation setup a fixed set of resources of a given type (e.g., CPU, memory) exists. We do not restrict the implementation to specific resources types. Although numerous types are provided it is possible to extend the set of types to meet the requirements of new simulation scenarios.

Task units access resources by means of resource requests. The request of a resource from the resource manager is modeled as an instantaneous event. Therefore, it can be realized as a method invocation at the corresponding resource manager object by the task unit object. The same holds for the other types of resource management messages.

The timing behavior is modeled within the task unit. After a task unit acquires the resources necessary for serving a packet, it starts to process the packets in the queue for incoming packets. For each packet to be processed an event is registered at the central scheduler. The time instant of that event coincides with the end of the processing of it.

The resource manager in turn administers the resources created for a specific simulation scenario. When multiple task units apply for the same resource, the resource manager resolves this conflict by assigning the resource to the task unit with the highest priority. Therefore, we are able to model resource contention scenarios in ns-3 in a unified way independent of the considered network.

Our resource management module described so far is independent of the other ns-3 modules. In order to model resource contention within computer networks it is desirable to combine this module with existing modules, e.g. protocol implementations. In the following we describe the approach chosen to integrate the general architecture into existing ns-3 modules.

As shown in Fig. 5, existing ns-3 classes like `Ipv4L3Protocol` are extended by multiple inheritance with our `TaskUnitAdapter` class to create an adapter object like `Ipv4ResourceManagement`. This adapter object references to a `TaskUnit` object which interacts with a `ResourceManager` object.

Such a child class like `Ipv4ResourceManagement` encapsulates the original ns-3 class `Ipv4L3Protocol` within the simulation setup. This way, it still uses and provides the same interfaces (e.g. `Receive()`, `Send()`) as the encapsulated, original ns-3 class. When a packet is passed to an instance of the newly introduced child class it is not processed directly but put into the queue for incoming packets of the corresponding task unit. Only after the modeled packet processing within the task unit, this packet is passed through the task unit method `FinishProcessing()` to the method of the parent class, here `Ipv4L3Protocol`, which handles the actual packet processing. Our extended ns-3 stack is illustrated in Fig. 6.

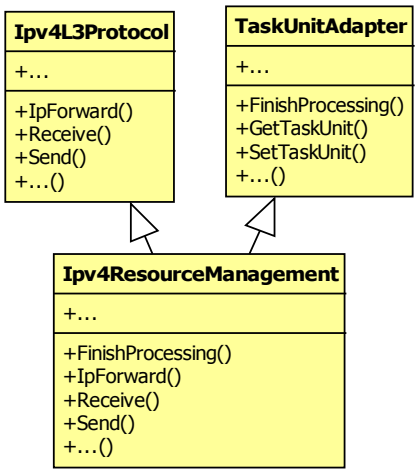


Fig. 5. Class Diagram of the `Ipv4ResourceManagement`

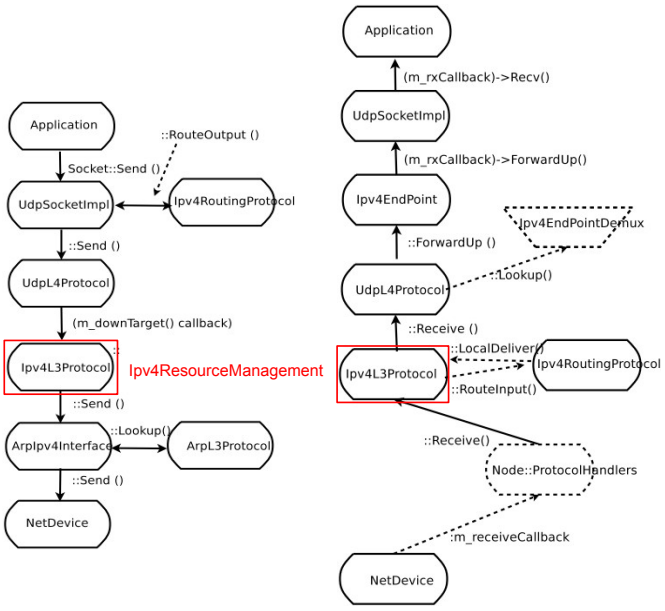


Fig. 6. Send and Receive Path of a Packet with Resource Management

The approach chosen allows for the integration of the resource management into various existing modules.

The usage of our resource management extension for ns-3 requires ca. 18 % extra computing time compared with the original ns-3 implementation in a worst case scenario with 64 B packet size, 20 Mpps packet rate and 4 cores. The resource management extension for ns-3 is open source software and publicly available under the GNU GPLv2 license for research, development, and use [32].

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 covered in RFC 2544 [13]. 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 lengths of at least 64, 128, 256, 512, 1024, 1280, and 1518 B. For our test case, this means performing multiple measurements at diverse frame rates and frame lengths and count the frames 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. Third, our network interface cards perform several techniques that influence incoming and outgoing packets (cf. Section III-C). 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 results in case of evenly distributed load. We also deactivated *Hyper-Threading*, which has no benefit in our case, as already one thread is able to max out the capacity of a 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 packets 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 [33] focus on building manifold traffic, i.e. in case of destination and source IPs, 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 (cf. Section III-A), uses a zero copy technique and is capable of filling our links using 64 B packets. This *pfsend* packet generator was used throughout 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.

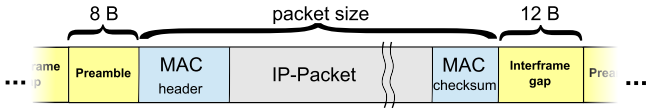


Fig. 7. Ethernet Frame Structure

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

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

Using Eq. (7) results in a maximum throughput of 14.88 Mpps that can be theoretically achieved with a 10 Gbps link and 64 B sized packets. We illustrated the Ethernet structure in Fig. 7. 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 with the help of our ns-3 *resource management* extension. The case study aims for the validation of our ns-3 extension. It is verified and validated based on real testbed measurements.

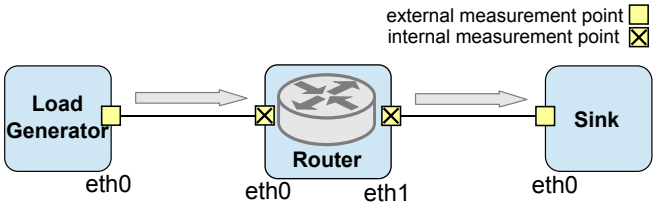


Fig. 8. Case Study Scenario with a Resource-Constrained Software Router

A. Scenario

The testbed and simulation scenario consists of a host *Load Generator* and a host *Sink* acting as end systems as well as a software router *Router* serving as device under test (Fig. 8). The load generator and the sink are connected via dedicated 10 Gbps Ethernet links to the router.

The data transmissions are uni-directional CBR traffic of 1, 2, 3, and 4 flows with constant packet sizes of 64, 128, 256, 512, 1024, and 1518 B from the load generator to the sink. The offered load of the load generator is greater than the maximal throughput $\hat{D}_{Router}$ of the software router under test. As a result, the maximum throughput achieved is depicted in dependence on the offered load to the software router.

1) Testbed Measurements: We implemented the software router using commodity server hardware. The system has been equipped with 16GiB 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 E3-1230 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 the latest *ixgbe* driver version (3.14.5) since we discovered its performance is significantly better than previous 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 flows with evenly distributed packet rates. These flows are crafted in a way that they are distributed to distinct cores by the RSS algorithm. This effectively means we utilize 1-4 cores in the router when producing 1-4 flows. Packet counters are implemented on the load generator and sink machines, depicted as the external measuring points in Fig. 8. The results of the throughput tests are displayed in Fig. 10.

2) Simulation Measurements: Our ns-3 resource management extension is applied to the router under test. The corresponding resource model for this case study is illustrated in Fig. 9. To process packets, the task unit $TU_{Packet\ Processing}$ has to request the resource manager RM_{Core} for a resource *core* of the CPU. If there are resources available in the resource pool RP_{Core} then the resource manager RM_{Core} allocates a core resource, e.g. C_1, to the task unit $TU_{Packet\ Processing}$ which starts to process the arrived packets from the incoming packet queue Q_{in}. The packet processing of the service unit $F_{Packet\ Processing}$ consumes simulation time corresponding to the required service time of the current packet (cf. Equation 6 in Section IV-B). Otherwise if there is currently no resource available, the task unit $TU_{Packet\ Processing}$ and also the arriving packets in the incoming packet queue Q_{in} have to wait until a *core* resource becomes available which additionally consumes simulation time.

In this case study, we assume that the cores of the CPU are the bottleneck. Therefore, only one resource type (namely the CPU cores) was considered but other resource types and intra-node effects (e.g. NIC Tx/Rx queues, cache misses) can be modeled to set up complex case studies which are hard to resolve analytically. Besides, the load generator and sink have no resource constraints, but the software router (DUT) possesses a limited number of four cores. The service time of a packet in the router depends on its packet size and the number of flows in the router (cf. Equation 6 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.

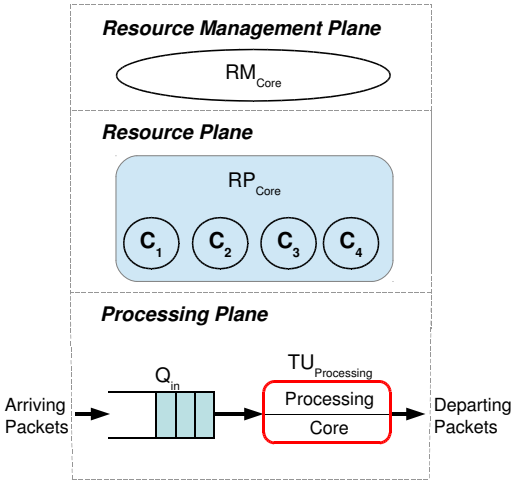


Fig. 9. Resource-Constrained Router Model

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 the model calibration because the CPU cores are not the bottleneck. This is the case, when applying data transmissions at a high level of offered load between the load generator and the sink. Here, the 10 Gbps Ethernet link becomes bottleneck instead of the CPU cores (cf. italic values in Table II).

In this case study, we just used the four measurement points for the calibration of the router simulation model. These calibration points are depicted as encircled points in Fig. 10(a) 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 I
MEASURED MAXIMUM THROUGHPUT ($\hat{D}_{meas}$) FOR VARYING NUMBER OF FLOWS (F) AND PACKET SIZES (S) USED FOR CALIBRATION OF THE MODEL PARAMETERS (a, a_0, b, b_0)

(a) Measurement Points

F	S [B]	$\hat{D}_{meas}$ [Gbps]
1	64	1.17460
1	512	7.41568
4	64	4.00731
4	128	7.07863

(b) Router Model Parameters

a	0.01135
a_0	0.00258
b	0.21766
b_0	0.06536

Based on these measurement values and according to our service time calculation of the router model (cf. Equation 6 in Section IV-B) the calibration parameters a, b, a_0, and b_0 of the router model can be derived. The values obtained are depicted in Table I(b).

C. Validation of Software Router Simulation Model

As mentioned in Section VI-B, the router model used by us 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 packet sizes.

Fig. 10(a) and 10(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 packet size in Byte and uses logarithmic scaling to basis 2. The y-axis represents the measured and simulated maximum throughput of the router in Mpps, and respectively in Gbps. The chart shows that the maximum throughput is not significantly dependent on the packet size because the routing table lookup overhead is equal for small and large packet sizes. However, the maximum throughput of a multi-core software router strongly depends on the number of flows because several flows can be distributed to multiple cores for parallel processing.

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 based on CBR traffic (cf. Section VI-A) do not show large variance. The mean deviation is ca. 0.25 % which indicates that our ns-3 extension is precise enough to produce realistic simulation results which is part of a successful model validation process. For the maximum throughput determination only the bold values of Table II are applicable because otherwise the 10 Gbps link is the bottleneck.

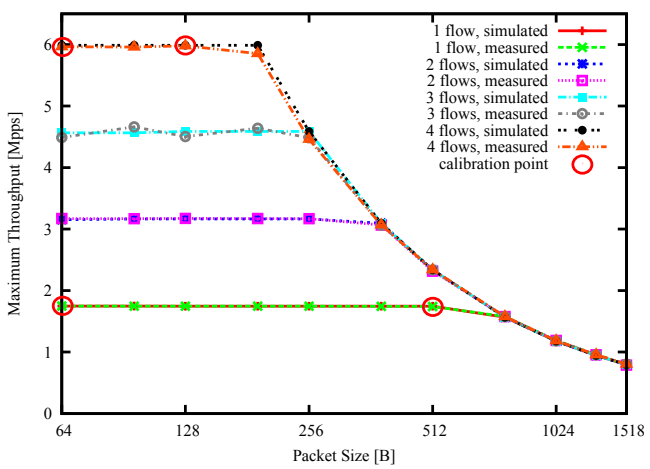
TABLE II
RELATIVE ERROR (Err) OF MEASURED ($\hat{D}_{meas}$) AND SIMULATED ($\hat{D}_{sim}$) MAXIMUM THROUGHPUT FOR VARYING NUMBER OF FLOWS (F) AND PACKET SIZES (S)

F	S [B]	$\hat{D}_{meas}$ [Mpps]	$\hat{D}_{sim}$ [Mpps]	Err [%]
1	**64**	**1.74792**	**1.74826**	**0.02**
1	**128**	**1.74541**	**1.74521**	**-0.01**
1	**256**	**1.74673**	**1.74521**	**-0.09**
1	**512**	**1.74241**	**1.74521**	**0.16**
1	*1024*	*1.18925*	*1.17926*	*-0.84*
1	*1518*	*0.79452*	*0.79746*	*0.37*
2	**64**	**3.17066**	**3.15458**	**-0.51**
2	**128**	**3.17262**	**3.16457**	**-0.25**
2	**256**	**3.16973**	**3.16457**	**-0.16**
2	*512*	*2.31526*	*2.33646*	*0.92*
2	*1024*	*1.19051*	*1.17926*	*-0.94*
2	*1518*	*0.78805*	*0.79746*	*1.19*
3	**64**	**4.48952**	**4.56622**	**1.71**
3	**128**	**4.50537**	**4.58717**	**1.82**
3	*256*	*4.48519*	*4.58717*	*2.27*
3	*512*	*2.33687*	*2.33646*	*-0.02*
3	*1024*	*1.18302*	*1.17926*	*-0.32*
3	*1518*	*0.80526*	*0.79746*	*-0.97*
4	**64**	**5.96327**	**5.98803**	**0.42**
4	**128**	**5.97857**	**5.98803**	**0.16**
4	*256*	*4.45911*	*4.58717*	*2.87*
4	*512*	*2.33979*	*2.33646*	*-0.14*
4	*1024*	*1.18407*	*1.17926*	*-0.41*
4	*1518*	*0.80664*	*0.79746*	*-1.14*

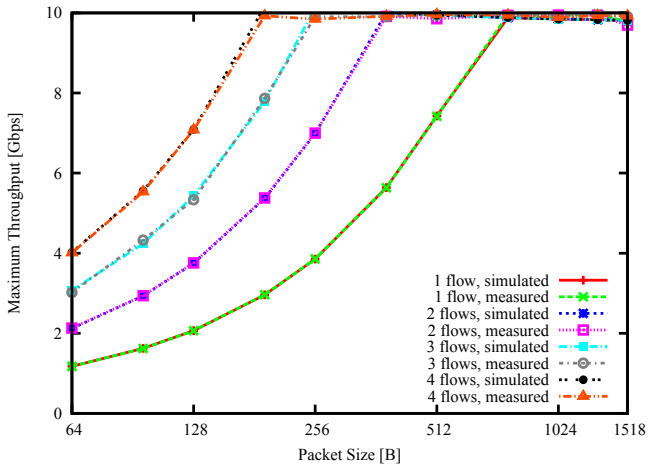
Although, we assumed a heuristic relation in the calibration of our resource management model for a quad-core router, (cf. Section IV-B), the validation experiments described here demonstrate that already simulation models which have been elaborated without much expenditure can lead to very realistic performance predictions if at least 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 future software routers. As the trend to larger number of CPU cores can be expected to continue instead of significantly

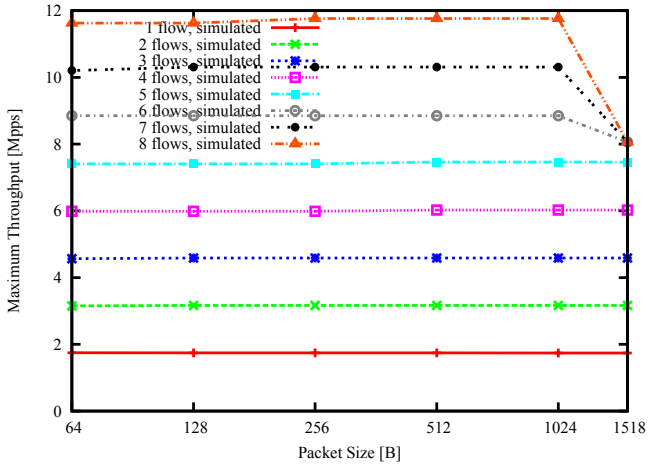


(a) Maximum Throughput in Mpps

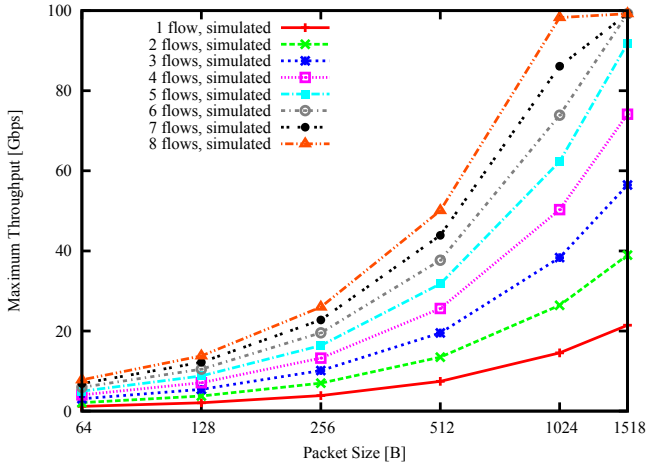


(b) Maximum Throughput in Gbps

Fig. 10. Simulation Results for the Maximum Throughput of the Modeled Resource-Constrained Software Router in Comparison to Real Testbed Measurements



(a) Maximum Throughput in Mpps



(b) Maximum Throughput in Gbps

Fig. 11. Simulation Prediction of the Maximum Throughput of a Modeled Resource-Constrained Software Router with 8 Cores and 100 Gbps Network Links

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. We model a software router based on a 8-core processor architecture which has to process 100 Gbps. A link speed of 100 Gbps may be supported by future NICs, but can already be implemented today using multiple 10 Gbps interfaces.

Fig. 11(a) and 11(b) illustrate the predicted maximum throughput by our simulation model in dependence of number of flows and packet sizes. The x-axis shows the packet size in Byte and uses logarithmic scaling to basis 2. The y-axis represents the simulated maximum throughput of the router in Mpps, and respectively in Gbps. This forecast shows that such a software router creates a bottleneck for 1 respectively 5 flows with packet sizes of 1518 B where this software router reaches its maximum throughput at 21.44 respectively 91.82 Gbps.

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 a pleasingly realistic model behavior. We used the model to predict how the trend of a growing number of CPU cores will affect the ability of 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 behaviour of a software router. In addition to our existing black-box measurements, we want to look into the routing software using code inspection and profiling. We hope to be able 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

This research has been supported by the *Deutsche Forschungsgemeinschaft* (DFG; German Research Foundation) as part of the *MEMPHIS* project (GZ: WO 722/6-1). We also would like to acknowledge the valuable contributions through numerous in-depth discussions from our colleagues Dr. Klaus-Dieter Heidtmann, Andrey Kolesnikov, Alexander Beifuß, Lothar Braun, and Thomas Schultz.

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 *22nd ACM Symposium on Operating Systems Principles (SOSP)*, October 2009.

[3] 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.

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

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

[6] R. Chertov, S. Fahmy, and N. Shroff, "A Device-Independent Router Model," in *27th IEEE Conference on Computer Communications (INFOCOM)*, April 2008, pp. 1642–1650.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward Predictable Performance in Software Packet-Processing Platforms," in *9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2012.

[12] 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.

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

[14] 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.

[15] "Quagga Routing Suite," http://www.nongnu.org/quagga/, (2013-05-15).

[16] "BIRD Internet Routing Daemon," http://bird.network.cz/, (2013-05-15).

[17] "Vyatta," http://www.vyatta.org/, (2013-05-15).

[18] 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.

[19] 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.

[20] "Open vSwitch," http://openvswitch.org/, (2013-05-15).

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

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

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

[24] 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.

[25] 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.

[26] 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.

[27] B. Mutnury, F. Paglia, J. Mobley, G. Singh, and R. Bellomio, "QuickPath Interconnect (QPI) Design and Analysis in High Speed Servers," in *19th IEEE Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, October 2010, pp. 265–268.

[28] 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.

[29] 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.

[30] 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.

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

[32] "MEMPHIS Project," http://www.informatik.uni-hamburg.de/memphis, (2013-05-15).

[33] 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, J. Schmitt, Ed., March 2012, vol. 7201, pp. 253–257.