How Do Multiple Network Cards Influence the Software Router Performance?

Torsten M. Runge[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
{runge|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 usage of cost-efficient and dynamically adaptable commodity PC hardware instead of specialized networking hardware has aroused strong interest in software routers. Testbed measurements as well as modeling and simulations of such system are important methodologies for finding bottlenecks in the packet processing to predict and optimize the performance. In this paper, we investigate the impact of the number of network cards with respect to the software router performance. We predict the performance of software routers based on a validated model which we obtain from real testbed measurements.**

***Keywords*— commodity hardware; packet processing; resource contention; network card; simulation model; ns-3; Linux NAPI**

I. Introduction

Commodity PC hardware has received several performance improvements. For instance, the trend towards many CPU cores as well as the multi-queue support of network interface cards (NIC) for parallel packet processing on multi-core architectures make off-the-shelf PC hardware interesting for the application as servers or routers. Furthermore, widely used open source operating systems such as Linux provide basic packet processing functionalities in software which can transform PC hardware into a so-called software router. The advantages of software routers are that they are more cost-efficient and more flexible to extend with new functionalities in comparison with special networking hardware such as hardware routers. However, hardware routers are often capable of higher packet processing performance and lower energy consumption. Nonetheless, in small to mid-range networks, such as home or campus networks, software routers represent an attractive alternative to expensive hardware routers.

The challenge for software routers is to process packet rates resp. data rates of million packets per seconds (Mpps) resp. multiple gigabits per second (Gbps) with no packet loss and acceptable packet latency. Therefore, the system internal components such as bus systems (e.g. PCIe, QPI) must provide the required data rates between the hardware components. Depending on the type of packet processing (e.g. routing, firewall, IPsec encryption) the software router must do several complex treatments per packet. In particular, the scaling of the number of NICs of software routers is important to compete with hardware routers.

For the optimization of the software router performance it is necessary to understand the packet processing steps in PC systems such as the Linux networking kernel as part of the operating system (OS) in detail. However, a software router is a complex system which consists of diverse hardware and software components interacting and influencing each other. Therefore, modeling and simulation may help to understand the complex interplay during the packet processing. Based on a sufficiently fine-grained model a complex system can be analyzed to find performance limiting factors and optimize the system accordingly to eliminate such bottlenecks. Finally, the model can be used to predict the performance behavior with respect to diverse scenarios which cannot be investigated in real testbeds.

In this paper, we measure and simulate the performance of software routers with respect to multiple network interface cards. Firstly, we conduct real testbed measurements with a Linux software router. Besides, we create a model for a software router with multiple network cards. This model is derived from our general concept for modeling of resource contention in resource-constrained nodes which is implemented as a resource management module for the widely used network simulator ns-3. The software router model is calibrated and validated with real testbed measurements. Based on that, we evaluate and predict the impact of multiple network cards on the performance of software router architectures.

The rest of the paper is structured as follows. Section II gives an overview of the state of the art in modeling, measuring, and implementation of software routers. Section III describes the procedures of the packet processing inside a Linux software router. In Section IV we describe our testbed which we used for the calibration of our software router model. Section V introduces and applies our general modeling approach to model a Linux software router with multiple network cards. Section VI presents a case study to illustrate the influences of multiple networks on the software router performance. The case study includes the calibration and validation of our simulation model based on the real testbed measurements. Finally, we conclude the paper and give an outlook to future work in Section VII.

II. Related Work

Many researches have shown that software routers are able to reach the performance of specialized networking hardware [1]–[3]. The RouteBricks project [1], [3] investigated the scaling of software routers based on commodity PC hardware. They applied modern NICs with multi-queueing support and exploit the parallel packet processing capabilities across multiple cores within a server as well as a software router cluster consisting of multiple servers. Bolla and Bruschi [4] identified the CPU to be the performance bottleneck in packet processing on commodity PC hardware which was also observed by RouteBricks [1] and us [5]. Therefore, small packets dominate the packet forwarding performance in software routers because a constant number of CPU cycles is needed per packet for the processing from the incoming port to the output port. Han et al. [2] investigated several improvements of the packet processing software by reducing the required number of CPU cycles per packet. Salim et al. [6], [7] investigated the Linux NAPI with respect to the reduction of the number of interrupts in high load situations. Salah et al. [8] evaluated and compared the performance of IP-packet forwarding of a Linux host equipped with three NICs.

Chertov et al. [9] proposed a router model which is able to predict the forwarding performance of arbitrary router devices based on calibration parameters retrieved from black box measurements. Salah et al. [10], [11] developed validated analytical models to estimate the performance of software based network hosts based on constant bit rate (CBR) and Poisson traffic. These models consider interrupt moderation mechanism of Linux and BSD with respect to the interrupt rate, the interrupt service routine (ISR) as well as packet processing times. We proposed a general and broadly applicable modeling approach [12] to investigate the node-internal resource contention during the packet processing. We implemented the modeling approach as a resource management extension module for the widely used network simulator ns-3. Based on that we evaluated the influence of the Linux NAPI with respect to packet latency [13].

Our best practice for measurements that we developed and proved in previous publications [5], [13]–[15] was influenced by the work from Bolla and Bruschi [16] and Dobrescu et al. [1], [17]. Both papers present detailed know how on performance measurements of software routers. For load generation we rely on *zsend*, which is based on the high speed network IO framework *PF_RING ZC* [18], which claims to produce CBR traffic. As zsend is a software-based load generator it cannot guarantee that the generated frames are accurately spaced. This is a general problem inherent to all software load generators [19]. We are actively working on an improved load generator to mitigate some of these issues by making use of specialized NIC hardware features (e.g. rate-limiting to avoid micro bursts) [20].

While we have studied the scalability of software packet processing based on the number of cores [5], the number of virtual machines [15], or by reducing the processing overhead [21] in the past, we are not aware of a detailed study of the performance overhead introduced by scaling the number of NICs on a packet processing system.

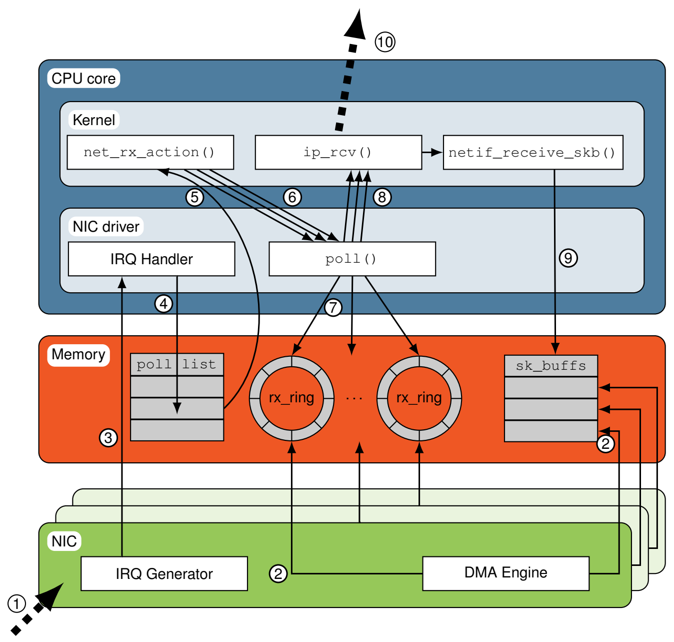


Fig. 1. Functional visualization of the processing path in a Linux router

III. Packet Processing in a Linux Software Router

As a first step to the analysis of Linux router scaling with the number of NICs we analyze the internals of packet processing. We sketch a simplified transmission path through the Linux router, which highlights aspects of scalability. Relevant elements are visualized in Fig. 1. For a more detailed description of packet processing functions in the Linux kernel we refer to our previous work [13], [22] in which we modeled the reception process in Linux as it is defined by the NAPI.

A. Packet Reception Path

Packets arriving at the NIC port (Fig. 1,①) are transferred to the memory via Direct Memory Access (DMA). The memory now contains the packet data and meta information which is stored in the `sk_buff`. A packet descriptor is an entry in the corresponding `rx_ring` which points to the respective `sk_buff`; ②. The NIC now triggers a hardware interrupt to the assigned CPU core; ③. The interrupt handler takes over and adds an entry to the `poll_list` that states that packets in the respective ring need to be processed; ④. The soft IRQ scheduler processes the soft IRQs which are processed by the `net_rx_action()`; ⑤.

The `net_rx_action()` function processes the entries of the `poll_list`. Batches of packets with a defined maximum batch size each get polled from the rings; ⑥. The actual poll happens via the NAPI function `poll()` that is defined as part of the NIC driver; ⑦. During that polling interrupts are deactivated. So higher loads result in a more polling-based packet processing while high interarrival times result in a mainly interrupt-driven packet reception. This essential function of the NAPI provides a flexible tradeoff between throughput and latency: at high load the NAPI avoids interrupts and aims for the maximum throughput, at low load the NAPI generates more interrupts to decrease latency. In order to avoid a CPU core that is locked by a never ending polling which would steal the CPU core capacity from other starving processes (e.g. user space programs that actually process the received packets) polling is stopped after a certain `budget` was used.

B. *From Reception to the Transmission*

In case of IP packets `ip_rcv()` is called for each packet; ⑧. An IPv6 packet would take a similar path (via `ipv6_rcv()`). For each packet in the polled batches of packets (stored in an `sk_buff`) the function `netif_receive_skb()` is invoked and MAC level information (e.g. VLANs) are processed; ⑨. The next steps can be categorized into prerouting, routing, and postrouting; ⑩. Packets can leave the routing path: e.g. if they are addressed to the host they get filtered out in the routing step and are directed on a path for local delivery. As all further processing of each single packet is independent from the NIC that received the packet, we do not discuss it in more detail here. The interested reader is referred to [23].

C. *Transmission Path*

After the routing decision, the packet gets prepared for transmission: A pointer to the `sk_buff` is placed in a `tx_ring` of the egress NIC port via the `dev_queue_xmit()` (ring selection) and the `hard_start_xmit()` (actual writing) functions. Locking of the ring only occurs if a `tx_ring` is not exclusively assigned to a CPU core. The NIC sends the packets of the Tx ring and triggers an IRQ after the successful transmission. Then the memory used for the sent packets must be freed from the Tx ring to store new packets for transmission. The Tx work limit defines the number of Tx descriptors which are cleaned in a single batch.

D. *Uniting Rx Flows from Different NICs*

Modern NICs come with support for multiple Rx and Tx rings that help to avoid any overhead for locking of the rings due to parallel processing. From the CPU's point of view the splitting and merging of parallel tasks is offloaded to the NICs. Thereby, the number of theoretically available `rx_rings` and `tx_rings` is some orders higher than the numbers of cores. Dedicating each CPU core to the traffic arriving at a certain NIC is inefficient when the load is not equally distributed among all NICs. For instance, one core may already be fully utilized while other cores are idle because traffic volumes in their NICs are low. Therefore, it is advisable to evenly distribute the incoming traffic to all CPU cores.

In most scenarios a core may have to serve packets from at least so many rings as there are NIC ports in the software router. Therefore, different packet paths have to be united to pass a single CPU core. This happens the same way packet paths coming in via different rings (and potentially just from one NIC) are merged: a requested poll task is placed in the poll list and processed in a round robin manner in the `net_rx_action()` function. By avoiding any overhead, when merging these ingress paths the NAPI ensures an important design objective: i.e. "robustness at any input rate and any number of input devices" [6]. Nevertheless, significant effects may e.g. be introduced by the increased number of available `sk_buffs` and `rx_rings` that lead to additional IRQ overhead.

E. *Side Effects*

As we have described the amount of ingress NICs that provide packets via the Linux NAPI should not affect the maximum packet rate of a software router significantly. In reality NIC vendors have implemented further techniques to increase the performance, e.g. Intel's interrupt throttling rate [24] that allows to delay interrupts by specifying a static or dynamic maximal rate of interrupts per second. As this maximal rate is ensured by delay and coalescing of interrupts in case of higher rates of ingoing packets these techniques are generally referred to as interrupt coalescing (IC). IC techniques work on a per-NIC-port-basis (or even per `rx_ring`) and therefore *may* introduce measurable effects of multiple NICs. As these are vendor specific features and not part of each NIC driver we do not analyze them further in this paper.

IV. Measuring the impact of multiple NICs

In this section we explain our measurements to quantify the overhead caused by distributing the load across multiple NICs in a software router. After defining our measurement goals, we derive a suitable measurement environment. We split our detailed description of the measurement environment in two parts, describing the load generator and the software router configuration.

A. *Measurement Goals*

The primary goal of our measurements is to assess the overhead of using multiple NICs for packet processing in a software router. Thus, we set up a Linux-based software router on a dedicated machine. We then test the routing performance of the software router in different load scenarios and measure the maximum packet rate of the device for each of the scenarios. The maximum achievable packet rate is suitable as an indicator for performance overhead, as it indicates the efficiency of packet processing. The higher the maximum observed packet rate is, the lower is the processing overhead of the software router.

As we are interested in both the resulting performance, and the root causes for overhead when using multiple NICs, we perform white-box measurements and look into our software router during the tests. First, we capture the CPU utilization on the software router in each of the scenarios. This allows us to compare the CPU load when using a different number of NICs and to calculate the number of cycles per packet necessary in each scenario, which is a direct indicator of the processing efficiency. A lower observed CPU load at a fixed packet rate indicates a lower processing overhead in this scenario.

In Section III we gave an overview of the packet reception and transmission process in the Linux NAPI. We present that the packet processing mechanisms heavily rely on interrupt handling and the avoidance of interrupts in the first place. Therefore, the number of interrupts generated by the NICs is a key parameter when analyzing the performance overhead by multiple NICs. We measure the number of interrupts on the Linux router to test this hypothesis.

B. Measured Scenarios

In order to measure the overhead of using additional NICs for packet processing, we create measurement scenarios where we keep all parameters stable except the number of NICs. Our hypothesis is that at a fixed amount of traffic the processing overhead changes with the number of NICs involved in the processing. Basically, this means that we keep the total amount of traffic sent to the software router stable, but split it in equal parts among a varying number of tested NICs. We derive three scenarios using 2 – 4 NICs as illustrated in Figure 2.

The scenario utilizing two NICs (Fig. 2(a)) utilizes bidirectional forwarding between two different NICs, splitting the load equally among both interfaces. The scenarios involving three and four NICs (Fig. 2(b) and 2(c)) involve a circular forwarding also splitting the load equally among all interfaces involved.

We test the maximum throughput of the software router according to RFC 2544 [25], which defines requirements for device benchmarking tests. Our software router, the *device under test* (DUT), is connected to a machine that generates packets for the DUT and captures packets coming back from the device.

Packet processing on the DUT is limited to one specific core in all our tests. For multicore-scaling we refer to our previous work that shows that the maximum possible packet rate scales linearly with a limited number of cores [5].

C. Software Router Configuration

Hardware We set up the software router on an off-the-shelf rack server. The machine comes equipped with a dual-socket Supermicro X9DRH-iTF Mainboard, two Intel Xeon E5-2640V2 CPUs with 8 cores each running at a maximum clock speed of 2.00 GHz. The mainboard comes with a dual-port 10 Gigabit Ethernet NIC on board (Intel Ethernet Controller X540). We added another dual-port 10 Gigabit Ethernet NIC (Intel 10 Gbit Network Adapter X540-T2) using the same controller. In total this makes four 10 Gbit/s NICs that we use for our tests. All four NICs are attached to the first of the two CPUs on board which is important because transferring packets from one CPU to the other via the dual-IO-hub would result in decreased performance [2].

CPU Settings When setting up and configuring the software, our goal is to make the behavior of the DUT predictable to get repeatable measurements and end up with meaningful results. We therefore try to eliminate side-effects that influence the performance of our software router. First, we disable the *Turbo Boost* and dynamic frequency underclocking features, which set the CPU clock speed in a non-predictable way. This pins the CPU clock speed to the maximum rate of 2 GHz during all tests. We also disable *Hyper-Threading*, which schedules virtual cores onto physical cores in an unpredictable way.

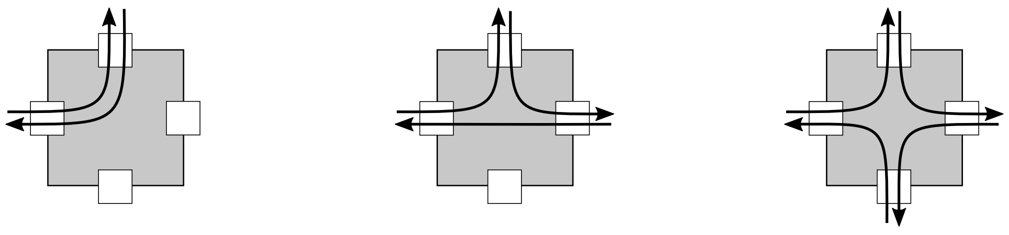
NIC Settings The NICs also need to be configured to facilitate reliable measurements. We disable Ethernet flow control, a feature that limits the network traffic in case one system is overloaded, because we also want to measure the DUT in overload situations. As multi-core measurements are out of scope, we configure only one `rx_ring` and `tx_ring` per NIC port. On top of that we disable the proprietary interrupt throttling feature offered by Intel, as explained in Section III. The Rx/Tx ring buffer sizes are set to 512 descriptors and offloading features such as *Large Receive Offload*, *Large Segment Offload*, and *Checksum Offloading* are disabled.

Software Our test systems run grml-Linux, a Debian-based live system distribution, running kernel version 3.7. As we want to measure the forwarding performance of the Linux-specific routing stack, we enable *ip_forward* and set static routes. As stated in the measurement goals, we configure the router to process all packets on one designated core. We already configured one `rx_ring` and `tx_ring` for each NIC port, now we bind the interrupts from all these queues to the same CPU core. We explicitly disable *irqbalance*, which has the goal to balance high-volume interrupts equally across all cores. Finally, we make sure the router can process incoming packets instantly without requiring any lookups, especially the ARP table is set manually before testing. In general we try to avoid any cross-traffic, e.g. by using statically assigned IP addresses.

System Performance Indicators We measure the CPU load and interrupt rate using `perf stat`, a lightweight tool to gather performance counters. To get reliable results, we delay launching perf stat for three seconds and let it end early before the end of a test run, to make sure that the system is fully loaded during the measured interval. We did not observe an influence of the measured throughput when running `perf stat`, in contrast to running a full-blown profiling setup, which would impact the system performance noticeably.

D. Measurement Device Configuration

We further limit the degrees of freedom in our measurement by using uniformly-sized packets with an Ethernet frame length of 64 Bytes, the minimum frame length supported by Ethernet. We previously showed that the performance of Linux-based software routers scales with the packet rate (number of packets per second) independent from the frame size as long as the link bottleneck is not reached [5].



(a) 2 NICs (b) 3 NICs (c) 4 NICs

Fig. 2. Overview of the traffic flow in the three measured scenarios. The gray box signifies the software router, while the arrows represent the traffic flows.

Load Generator To make use of our 10 GBit/s NICs we use the *zsend* software load generator, based on the packet processing framework *PF_RING ZC* [18]. We configure zsend to generate a constant bit rate (CBR) stream of uniform-sized UDP packets. As we want to distribute the load equally among several interfaces, we run zsend for each outgoing interface with a fraction of the specified packet rate, so that the total load generated sums up to the target packet rate. Additionally, we set the destination IP address, specifically for each instance of zsend, so that we can control the output interface on the DUT.

Packet Counters Our NICs feature hardware packet counters that accurately count the number of packets transmitted and received. We periodically poll these hardware registers to keep track of the number of packets sent and received on each network interface.

V. Performance Evaluation with Simulations

The methodology of simulations is often used in education and scientific studies because the setup of real testbed which implies complex configurations of the networking devices and links is often expensive and time consuming. Instead, researchers can use simulators as a cost-effective approach to design, validate, and analyze new ideas and optimizations in a controlled and reproducible manner.

A. Modeling of Resource Management in ns-3

To model a Linux software router, we apply our general modeling approach for intra-node resource management in resource-constrained network nodes [12]. Among others, this modeling approach introduces the terms resource manager, task unit, resource, and resource pool. A *resource manager* is an abstraction of the fundamental functions of the operating system (e.g. resource allocation, scheduling). All resources of a specific type are located in a resource pool which is managed by the resource manager according to a specific scheduling strategy (e.g. round-robin, priority scheduling). A *task unit* (TU) models an OS process or thread which requires specific resources like a CPU core or memory to be executed. Therefore, if a task unit has to process packets, it has to request the corresponding resources. For instance, in case of the Linux NAPI this refers to an IRQ after the reception of a packet. If the requested resource is available, then the resource manager sends a reply to the task unit which corresponds to the scheduling of an Rx ring by the NAPI.

B. Modeling of a Linux Software Router

Our Linux software router model consists of multiple network interface card ports ($NIC_0, \ldots, NIC_n$) and multiple CPU cores ($C_0, \ldots, C_k$) as depicted in Fig. 3. In the network simulator ns-3 each NIC port is represented by a specific *Net Device*. The *Net Device Classifier* assures a one-by-one mapping between net devices and its corresponding net device task units of the ns-3 resource-management module to model the ingoing NIC port ($TU\,NIC_{in}$). A net device task unit models the behavior of the network card controller. For instance, the multi-queue feature of modern NICs is modeled in that way that the incoming traffic may be classified based on specific packet attributes into one of the incoming queues (aka. Rx rings) of the successor task units to achieve traffic load balancing or prioritization of specific traffic.

In a Linux software router, these successor task units model behavior of the interrupt moderation of the Linux NAPI. There may be multiple Rx rings which are served by one NAPI thread pinned to a specific CPU core. This NAPI thread is modeled as multiple NAPI task units ($TU\,NAPI$). Each NAPI task unit possesses a dedicated incoming queue which models a specific Rx ring. Therefore, to model a Linux software router with $(n+1)$ NICs and $(k+1)$ CPU cores we need $(n+1)\cdot(k+1)$ NAPI task units. These NAPI task units compete for the same shared CPU core resource. Thus, NAPI task units which are pinned to the same core cannot be processed simultaneously. This refers to the fact that a NAPI thread in the real system can only serve one Rx ring at a certain time.

Depending on the type of the configured packet processing of the software router, diverse task units for each packet processing step may follow. For instance, in the case that the software router models a VPN gateway then a processing task unit ($TU\,Process$) is included which represents the packet processing step for IPsec encryption. In consequence, each additional processing task unit requires CPU resources ($C_0, \ldots, C_k$) which leads to an increase of the required CPU cycles per packet. After the actual packet processing, the packet is enqueued in the corresponding task unit which models the outgoing network interface card ($TU\,NIC_{out}$). There is a one-by-one mapping between the task units for the outgoing NIC ports and the corresponding ns-3 net devices. Finally, the packet is enqueued in the transmission queue of the outgoing net device and the processing of the native ns-3 core continues.

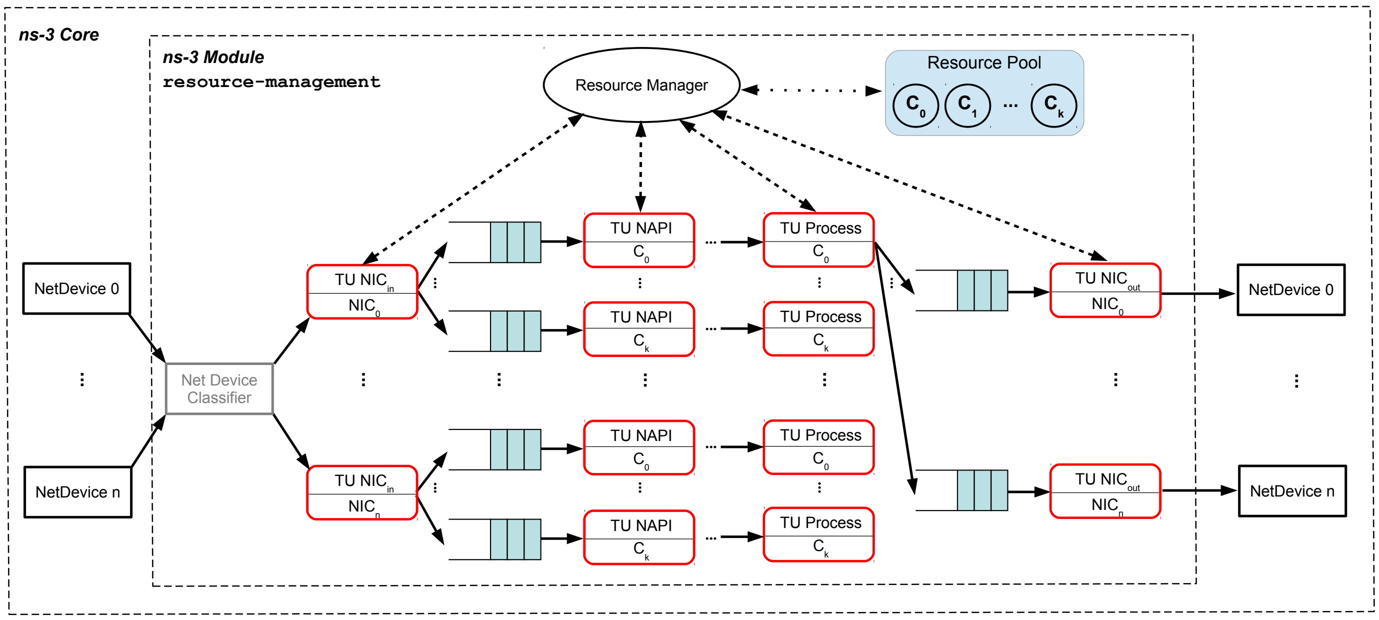


Fig. 3. Node model of a Linux software router with multiple network cards

C. Model Assumptions and Limitations

The software router model is subject to the following assumptions and limitations.

- The simulation model assumes that the CPU constitutes the bottleneck during the packet processing. Furthermore, we assume a linear scaling in the throughput when applying multiple CPU cores in parallel. This behavior was also shown by us [5] and other researchers [1].
- The simulation model considers modern NICs with multi-queue support which means that each CPU core has at least one Rx/Tx ring per NIC. This assures that every CPU core is able to exclusively access its dedicated Rx/Tx ring without resource contention. Thus, the modeling of locking mechanisms to access the Rx/Tx rings can be omitted. Besides, the processing of a specific packet always remains at the same CPU core to avoid cache misses which aligns with common recommendations for parallel packet processing in multi-core PC systems [1].
- We assume that the outgoing link is not the bottleneck. Therefore, the model does not implement any queuing disciplines (qdisc) at the egress NIC. This is in contrast to many network simulators like ns-3 [26] which assume the outgoing link to be bottleneck.
- The model neglects concurrent processes or threads which may lead to context switches. However, concurrent processes can be easily modeled by introducing additional task unit which apply for the same shared CPU resource. Furthermore, the OS scheduling overhead for managing multiple processes is neglected.
- Additional latencies introduced by other hardware components (e.g. DMA, PCI) are omitted but can be modeled with the help of our ns-3 resource-management extension to set up more fine-grained case studies.

VI. Case Study: Scaling of Linux Software Routers

In this section we evaluate the influence of the number of network cards with respect to the packet processing performance of a Linux software router. Firstly, we measure the performance of a Linux software router in real testbed experiments. Based on these testbed measurements we calibrate and validate our Linux software simulation model which is based on our *resource management* extension module for the network simulator ns-3.

A. Case Study Scenario

1) Network Topology: The case study scenario consists of a specific number of end systems connected to a Linux software router which acts as the device under test. The number of end systems is varied to investigate the influence of multiple network cards (cf. Fig. 2).

2) Load Generation: For all conducted measurements the load generation is based on the following conditions.

- All traffic is mapped to only one CPU core of the Linux software router because earlier research showed that the performance of multi-core CPUs linearly scales with the number of CPU cores [2], [3], [5], [27].
- The offered load is uniformly distributed between the number of NICs of the software router.
- All measurements are conducted with an Ethernet frame length of 64 B, because we have already shown that the packet size has no significant impact on the performance in case of IP routing. Furthermore, with small packet sizes we prevent that the network interface cards become the bottleneck at high offered loads.

3) Linux Software Router Configuration:

- The considered CPU core of the software router runs at a clock frequency of 2 GHz. The Intel features turbo boost and hyper-threading are disabled to prevent the system from tuning the clock frequency because this would evidently influence the performance of the software router.
- All used network cards are 10 Gbps Ethernet NICs.
- The interrupt moderation of the Linux software router relies on the NAPI. Therefore, we disabled the proprietary Intel NIC feature ITR (Interrupt Throttling Rate) [28], which is an additional interrupt moderation feature to avoid side effects.

As explained in Section IV we measured the packet forwarding performance of a real Linux software router. These testbed measurements aim to serve as calibration points for our software router model described in Section V. We therefore use the same parameters for all three scenarios during measurement and simulation. All measurements were conducted with CBR traffic consisting of uniform UDP packets with an Ethernet frame length of 64B. We present the results of these measurements in Figure 4(a) and Figure 4(b).

B. Model Calibration and Validation

Our Linux software router model was calibrated and validated for IP packet processing based on real testbed measurements of a Linux Software Router. The testbed measurements were conducted based on constant bit rate (CBR) traffic. The measured as well as the calibrated values for the simulation model are shown in Fig. 4(a) and 4(b). We measured a maximum throughput of the Linux software router of 0.9 Mpps for a single CPU core running at 2 GHz which relates to a service time of 1111 ns resp. 3667 CPU cycles per packet.

Through internal measurements by profiling the Linux networking stack with the tool *perf*, we measured the effort for the packet processing in detail. We observed that the service time is nearly constant and independent of the packet size because each IP packet requires a similar number of CPU cycles for the IP header processing (e.g. routing table lookup). The profiling results show that ca. 98.5 % of these 1111 ns are consumed for the polling of packets which consists of 86 % resp. 940 ns for the packet reception and actual packet processing (e.g. NAPI, IP processing) as well as 14 % resp. 154 ns for the packet transmission (e.g. Tx ring cleaning).

Furthermore, based on the testbed measurements we estimate the cost for an ISR with ca. 3400 ns which arises if an incoming packet causes an IRQ. Besides, we use a NAPI poll size of 64 and a Tx work limit of 256 which refer the defaults of the Linux NAPI. Further details of the derivation of the model calibration parameters are presented in [13].

The simulation results coincide with the testbed measurement values very well which indicates that our simulation model is valid and has been implemented correctly (Fig. 4). For the CPU utilization, a derivation offset of 10 to 20 % occurs for offered loads between 0.2 and 0.4 Mpps due to additional processing costs which are not considered in the simulation model. Furthermore, the applied CBR traffic cannot be precisely guaranteed with the load generator in the testbed. This is in contrast to the load generation in the simulator which is able to assure accurate packet interarrival times.

C. Simulation Results

The case study scenario is modeled based on the network simulator ns-3 as well as our ns-3 extension module *resource-management* [12]. The end devices and the software router are represented as dedicated ns-3 nodes. For the load generation, on each end system a UDP client as well as a UDP server application is installed. The UDP clients generate traffic with a constant packet size of 64 B according to a Poisson stream. The Linux software router possesses limited resources, thus our ns-3 resource management module must only be deployed to the software router.

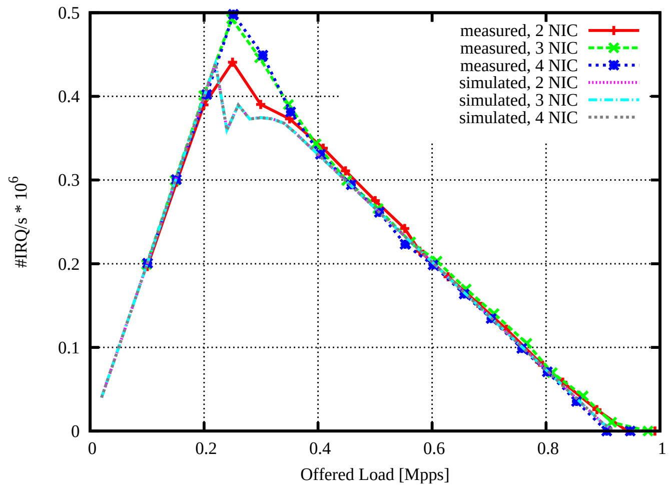
In contrast to the testbed measurement in the model calibration, the offered load is based on Poisson traffic with exponentially distributed interarrival times. Thus, the offered traffic is more realistic because short packet bursts as well as short idle times with no packet arrivals occur. Besides, for a specific offered load the overall IRQ rate is smaller with Poisson traffic in comparison to CBR traffic because when multiple packets arrive in a short burst only one IRQ is triggered which also reduces the CPU utilization. We analyzed the IRQ rate, the CPU utilization, and the mean packet latency for a Linux software router with different numbers of NICs.

1) IRQ Rate and CPU Utilization: The IRQ rate is a measure for the IRQ overhead in number of IRQs in a specific time interval triggered by the NICs. The CPU utilization is a dimensionless metric for the usage of the CPU as the relation between the busy time and the total time of observation.

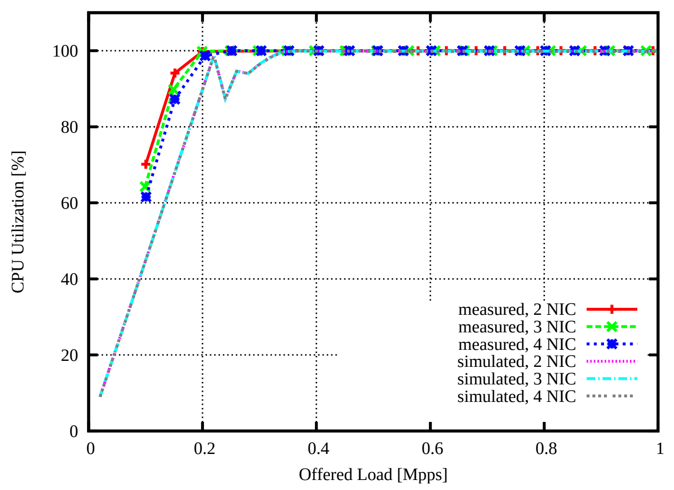
Fig. 5(a) and Fig. 5(b) show the offered load in million packets per second (Mpps) on the x-axis. Fig. 5(a) illustrates the IRQ rate in number of million IRQs per seconds on the y-axis whereas Fig. 5(b) represents the corresponding CPU utilization in percentage on the y-axis. Each of the lines in both figures represents a different configuration of the software router with respect to the number of NICs.

For low offered loads below 0.3 Mpps the CPU core is idle most of the time. Here, the Linux NAPI behaves interrupt-driven because almost every incoming packet causes an IRQ and is directly processed. Thus, when the offered load increases the IRQ rate as well as the CPU utilization linearly increases. However, the higher the offered load becomes the bigger is the difference in the IRQ rate and CPU utilization for the different number of NICs. For instance, at an offered load of 0.3 Mpps we observe an IRQ rate respectively CPU utilization with 2 NICs of $0.27 \cdot 10^6$ IRQ/s resp. 79 % whereas with 4 NICs the values are $0.31 \cdot 10^6$ IRQ/s resp. 86 %.

For an offered load higher than 0.3 Mpps, the IRQ rate decreases because there are often packets in the Rx rings of NICs and thus the Linux NAPI disables the IRQs for the corresponding Rx/Tx rings. Therefore, the Linux NAPI often works in poll-driven mode which saves IRQs. Thus, the rising of the CPU utilization also diminishes. When the offered load reaches the maximum throughput of the CPU core at ca.

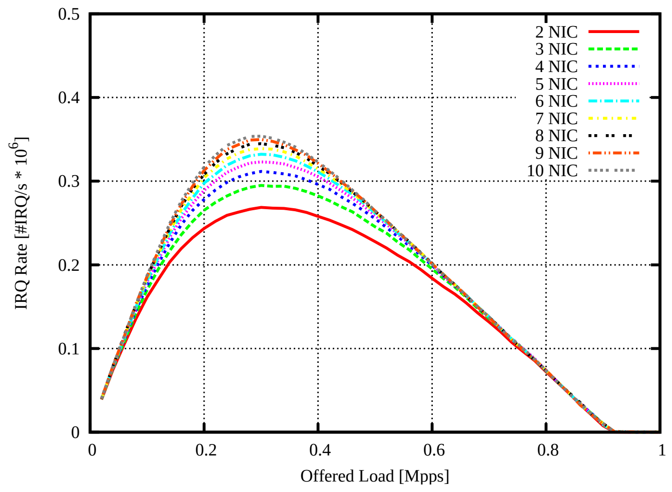


(a) IRQ Rate

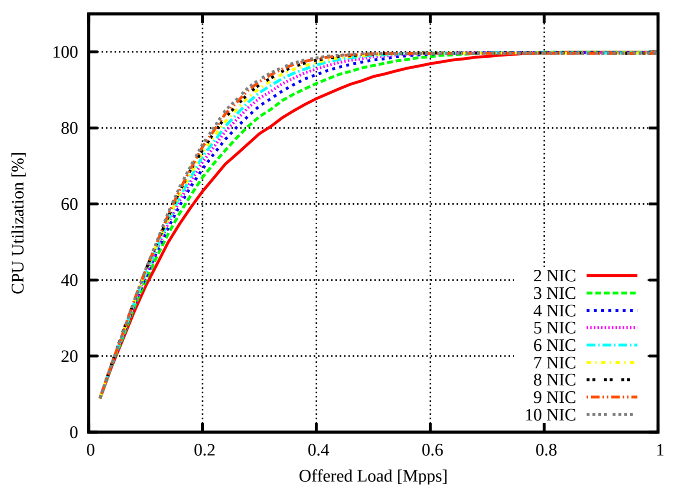


(b) CPU Utilization

Fig. 4. Measured and simulated IRQ rate and CPU utilization for different numbers of NICs based on CBR traffic used for model calibration



(a) IRQ Rate



(b) CPU Utilization

Fig. 5. Simulated IRQ rate and CPU utilization for different numbers of NICs based on Poisson traffic

0.9 Mpps, then all IRQs of the Rx rings are disabled to use the saved CPU cycles for the actual packet processing.

We conclude that with increasing offered loads the CPU core becomes saturated earlier with a higher number of NICs because at the same offered load each NIC triggers a separate IRQ for each packet. In contrast, with a smaller number of NICs at the same offered load, an IRQ serves multiple packets which leads to less IRQ-induced CPU utilization.

2) *Packet Latency:* The packet latency represents the delay which a packet incurs during its traversal through the software router. The packet latency consists of waiting and services times in several system components. In general, the waiting time of a specific packet depends on the number of packets prior to that packet in the waiting queue which is here an Rx ring. In the case of IP routing, the service time of a packet is constant (cf. Section VI-B). The packet latency is dominated by the waiting and service time at the CPU bottleneck.

Fig. 6 shows the offered load in Mpps on the x-axis and the simulated mean packet latency in microseconds on the y-axis. The mean packet latency is stated with 95 % confidence interval. Each of the lines represents a different configuration of the software router with respect to the number of NICs. The mean packet latency exponentially increases when the offered load increases. For offered loads higher than the maximum throughput of 0.9 Mpps of this router configuration, the mean packet latency is no longer well-defined. This is because arriving packets often hit a full incoming packet queue and must be dropped. For the same offered load, the mean packet latency is bigger the higher the number of NICs is. This is because with multiple NICs more IRQs must be handled which CPU cycles cannot be used for the actual packet processing which leads to higher waiting times. Consequently, with a higher number of NICs the full CPU utilization of 100 % is already reached for lower offered loads (cf. Figs. 4(b), 5(b)).

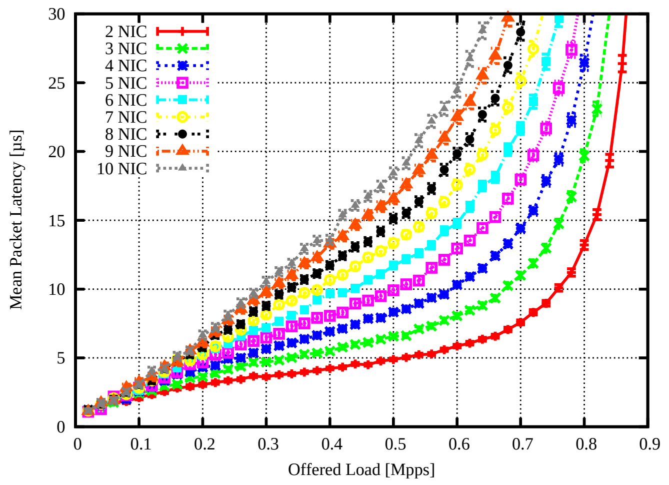


Fig. 6. Simulated mean packet latency for different numbers of NICs

VII. Summary and Outlook

We investigated the impact of multiple NICs for the performance of Linux software routers. We proposed a model of a Linux software router which we calibrated and validated based on testbed measurements. We showed that the Linux NAPI is able to scale with multiple NICs. However, the number of NICs have an impact on the performance, especially for the mean packet latency due to the additional IRQ overhead.

In future, we plan to further refine our model which includes the consideration of the latency caused by other system components. Besides, we are implementing a Linux NAPI extension for QoS-aware packet processing in 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 would like to acknowledge the valuable contributions through numerous in-depth discussions from our colleagues Alexander Beifuß, Paul Emmerich, Sebastian Gallenmüller, Andrey Kolesnikov, and Dr. Klaus-Dieter Heidtmann.

References

[1] 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.

[2] S. Han, K. Jang, K. Park, and S. Moon, "Building a Single-Box 100 Gbps Software Router," in *IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, May 2010, pp. 1–4.

[3] K. Fall, G. Iannaccone, M. Manesh, S. Ratnasamy, K. Argyraki, M. Dobrescu, and N. Egi, "RouteBricks: Enabling General Purpose Network Infrastructure," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 1, pp. 112–125, 2011.

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

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

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

[7] J. H. Salim, "When NAPI Comes to Town," in *Linux Conference*, 2005.

[8] K. Salah and M. Hamawi, "Performance of IP-Forwarding of Linux Hosts with Multiple Network Interfaces," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 452 – 465, 2013.

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

[10] K. Salah, K. El-Badawi, and F. Haidari, "Performance Analysis and Comparison of Interrupt-Handling Schemes in Gigabit Networks," *Computer Communications*, vol. 30, no. 17, pp. 3425–3441, 2007.

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

[12] T. M. Runge, B. E. Wolfinger, S. Heckmüller, and A. Abdollahpouri, "A Modeling Approach for Resource Management in Resource-Constrained Nodes," *Journal of Networks*, vol. 10, no. 01, pp. 39–50, 2015.

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

[14] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "A Study of Network Stack Latency for Game Servers," in *13th Annual Workshop on Network and Systems Support for Games (NetGames'14)*, Nagoya, Japan, Dec. 2014.

[15] ——, "Performance Characteristics of Virtual Switching," in *IEEE International Conference on Cloud Networking (CloudNet)*, Luxembourg, October 2014.

[16] 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.

[17] 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.

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

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

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

[21] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of Frameworks for High-Performance Packet IO," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2015)*, 2015.

[22] T. M. Runge, A. Beifuß, and B. E. Wolfinger, "Low Latency Network Traffic Processing with Commodity Hardware," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2015.

[23] D. Raumer, F. Wohlfart, D. Scholz, P. Emmerich, and G. Carle, "Performance Exploration of Software-based Packet Processing Systems," in *Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und verteilten Systemen, 8. GI/ITG-Workshop MMBnet*, September 2015.

[24] P. Emmerich, D. Raumer, A. Beifuß, L. Erlacher, F. Wohlfart, T. M. Runge, S. Gallenmüller, and G. Carle, "Optimizing Latency and CPU Load in Packet Processing Systems," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2015.

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

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

[27] 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.

[28] Intel Corporation, "Interrupt Moderation Using Intel GbE Controllers," April 2007.