Europa: Efficient User Mode Packet Forwarding in Network Virtualization

Yong Liao[†]*, Dong Yin*[‡]*, and Lixin Gao*[†]
University of Massachusetts at Amherst (USA)[†]*, Northwestern Polytech University (China)*[‡]

Abstract

Network virtualization provides the ability to run concurrent virtual networks over a shared substrate. However, it is challenging to design such a platform to host multiple heterogeneous and often highly customized virtual networks. Not only minimal interference among different virtual networks is desired, high-speed packet forwarding is also required. This paper presents Europa, a virtual network platform uses *Efficient UseR mOde Packet forwArding*, which supports high-speed and highly customizable virtual networks. Our platform adopts lightweight OS-level virtualization to slice a physical server into virtual machines. The data plane of a virtual router runs in an isolated virtual machine so as to safe for customization. We design a new user mode packet processing scheme for virtual routers hosted in Europa to achieve high speed forwarding. Experiments show that an Europa virtual router can be four times faster than conventional user mode software router.

1 Introduction

Network virtualization has been proposed as a powerful approach to facilitate testing and deploying network innovations over a shared substrate. In a network virtualization infrastructure, concurrent virtual networks can be created so that it is possible to independently deploy and experiment with new innovations. Virtual networks should be isolated from each other to minimize the interference among them.

However, it is challenging to build a shared substrate that can support multiple concurrent virtual networks. The intrinsic heterogeneous nature of network innovations requires that a virtual network must be highly flexible and customizable. It is often required to tune various aspects of the virtual networks. For example, a virtual network may be created to test new routing protocols, and therefore, its control plane needs to be customized. One may also experiment with new packet forwarding functions in a virtual network, such as queuing schemes or new addressing mechanisms, which cannot be realized without a customized data plane. In addition to the flexibility requirement, to experiment and test network innovations in a realistic environment, and more importantly, to attract long term deployment of new applications, a network virtualization platform should provide good data plane performance as well. It is desirable that the overhead of virtualization is minimized, so that the data plane performance of the platform can closely approach the full potential of the underlying hardware.

Achieving both high degree of flexibility and high performance is challenging. To guarantee the isolation between virtual networks so as to provide the flexibility to do customization, both control plane and data plane of a virtual network should run in the unprivileged domain of the hardware, which can introduce overhead. Although this overhead may not be an issue for control plane functions, it can largely impact data plane performance. For example, the VINI platform [1] provides high degree of flexibility by running virtual network data planes in operating system user mode, but the packet forwarding speed of VINI is much slower than what the hardware can potentially achieve. Running the data plane of a virtual network in OS kernel mode, as what Trellis [2] does, can achieve much better performance. However, Trellis is limited in its ability to customize data plane in a virtual network due to the constraint imposed by the forwarding function provided by the kernel.

In this paper, we explore how to build a network virtualization platform that can achieve high degree of flexibility without sacrificing data plane performance. We propose *Europa*, a virtual network platform built from commodity hardware. Europa puts flexibility as its first design goal. Hence, the data plane of a virtual network hosted in Europa has to run in a virtual machine and essentially in OS user mode, so that a virtual network can be granted the full control of its data plane. We design a new user mode packet forwarding scheme for Europa, which can achieve high forwarding speed. Experimental results show that although an Europa virtual network runs its data plane in OS user mode, it can achieve close to the best known software router data plane performance. Europa achieves high forwarding speed by adopting two mechanisms, i.e., sharing packet buffer and polling packet state.

First, unlike the conventional ways of forwarding packets in user mode, our scheme uses shared memory to store packets and eliminates the overhead of copying packets between user space and kernel space. Although this idea has been exploited in designing zero-copy I/O mechanism in operating system [3], our paper is the first one to apply this idea in building network virtualization platforms with both high-degree of flexibility and high packet forwarding speed, which is of practical value as network virtualization is attracting increasing interests from both research community and industry.

Second, our scheme also avoids the overhead of invoking system calls by letting a user mode virtual router and OS kernel independently poll the state of a packet, which is an important factor differentiating our work from existing work in operating system area. As we will see later in the paper, avoiding the overhead of invoking system calls is important for an Europa virtual router to forward packets at high speed, because using system calls to interact between user mode process and kernel introduces considerable overhead. Note that when there is no packet to forward, polling for packets could consume lots of CPU resource without forwarding any packets. However, even a virtual router uses all its CPU cycles in polling, this does not impact the resource isolation among virtual routers because the server virtualization mechanism ensures a virtual router not to exceed its CPU resource quota. To save CPU cycles when a virtual router has no packet to forward, an adaptive polling mechanism can be adopted, which slows down the polling frequency when the incoming packet rate is slowed.

The rest of this paper is organized as followings. In section 2, we examine different types of software routers running in commodity hardware and study why the conventional user mode packet forwarding has degraded performance. Section 3 presents our Europa platform that uses efficient user mode packet forwarding. The experimental evaluation results are presented in section 4. Section 5 concludes this paper.

2 Packet Forwarding in Software Routers

A software router, like the widely used Click router [4], can run in both OS kernel mode and user mode. Running Click in kernel mode achieves reasonably good forwarding speed [5,6]; user mode Click, on the other hand, forwards packets much slower. In this section, we examine the packet forwarding procedures of kernel and user mode software routers. The study can help us to understand why conventional user mode software router has slow forwarding speed.

2.1 Packet Forwarding Procedures

When a packet is received by the NIC (Network Interface Controller) hardware, a kernel mode software router usually starts a DMA process to transfer the packet into a buffer in kernel space and processes the packet in the "in-place" manner. After kernel mode router decides how to forward that packet, it transfers the packet to the outgoing NIC via DMA. We test the forwarding speed of kernel mode Click running in a commodity PC, which has a 2.66GHz dual-core CPU and Gbit PCIe NICs. In our tested machine, the maximum forwarding speed is about 1050K packet per second (pps) for 64-byte packets.

In Linux platform, user mode Click router uses the so-called PF_PACKET socket to interact with the kernel to send and receive packets. When a packet is ready in NIC, the kernel uses DMA to transfer it from NIC to main memory and attaches that packet to the kernel buffer associated with the PF_PACKET socket. When the packet-receiving task is scheduled, Click calls the *recvfrom()* system call to copy the entire packet into a user space buffer. After processing the packet, Click invokes the *send()* system call to copy the packet back to kernel and kernel sends the packet out via NIC. We test the forwarding speed of user mode Click using the same machine in testing kernel mode Click. The maximum forwarding speed is about 230 Kpps.

2.2 Causes of Slow User Mode Forwarding

We see that the major difference between packet forwarding in kernel mode Click and user mode Click is two-fold. For user mode Click to forward a packet, (i) that packet needs to be copied between the kernel space and user space twice; (ii) user mode Click needs to invoke two system called to interact with the kernel. In the following, we quantify these two types of overhead.

Memory copying: We measure the CPU cycles needed to copy data between user space and kernel, by using the "time stamp counter" of Intel CPU [7]. The results are shown in Table 1. For 64-byte packets, it takes about 140 cycles to copy a packet from user space to kernel and 160 cycles to copy a packet from kernel to user space.

packet size (bytes)	64	128	256	512	1024	1500
copy_to_user	162	188	239	302	442	575
copy_from_user	140	157	200	259	388	507

Table 1: CPU cycles used to copy packets.

System call: We measure the overhead of invoking the *recvfrom()* and the *send()* system calls used in user mode Click. Our measurement (Table 2) shows that for the user mode Click to forward one packet, the overhead of using system calls is about 6,400 CPU cycles.

system call	send()	recvfrom()
CPU cycles	3000 cycles	3400 cycles

Table 2: CPU cycles consumed in invoking system calls.

To forward one packet in user mode Click, there is a total of 6,700-cycle extra overhead as compared with forwarding packet in kernel mode Click. In our tested machine with a 2.66 GHz CPU, the extra 6,700 CPU cycles would limit the forwarding speed of user mode Click to no more than 400 Kpps for 64-byte packets.

Our study indicates the directions of efficient user mode packet forwarding. First, user mode software router should avoid using system calls to interact with OS kernel. Although system call is the most common way to interact between user and kernel space, the overhead is too expensive for achieving high speed packet forwarding. Second, it is desirable to copy packets as less as possible. Ideally, a packet should be processed in the "in-place" manner once it is in the main memory.

3 Europa System Design

Having identified the main causes for slow packet forwarding in conventional user mode software router, in this section we present Europa, which adopts an efficient user mode packet forwarding scheme to facilitate high speed packet forwarding in virtual networks.

3.1 Basic Ideas

Firstly, to avoid copying packets between user space and kernel, Europa adopts a mechanism widely used in OS inter-process communication. That is, for each virtual router hosted in an Europa server, we have a buffer to store all its packets. The buffer is shared between the virtual router and the OS kernel, so that the virtual router running in user mode can directly process packets stored in the buffer without copying them back-and-forth between user space and kernel space.

Secondly, to avoid using expensive system calls, Europa adopts an asynchronous model for a virtual router and the kernel to access their shared buffer. Both the virtual router and the kernel independently monitor the "state" of a packet stored in the shared buffer, e.g., they poll the state of a packet. If either the user mode virtual router or kernel notices that a packet is ready to be taken over, the user mode virtual router or kernel starts to process the packet.

3.2 Europa Architecture

Figure 1 depicts the basic architecture of Europa. An Europa server is sliced into *virtual routers* using lightweight OS-level virtualization mechanism [8]. Each virtual router has a data plane running inside itself. For example, Figure 1 shows an Europa server hosting two virtual routers. An Europa server runs a module, referred to as the EuropaKM, inside its OS kernel. The EuropaKM receives packets from NICs of the Europa server and classifies the packets to their virtual routers. After packets are processed by the virtual routers, EuropaKM will send them out to NICs.

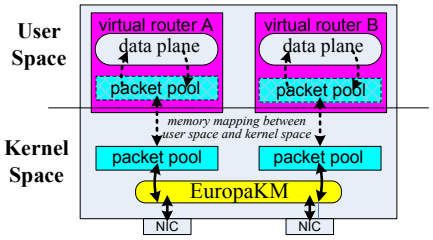


Figure 1: Europa architecture.

Sharing packet buffer: For each virtual router, a memory buffer, called the *packet pool*, is allocated by EuropaKM to store all packets belong to that virtual router. The packet pool is shared to the user mode virtual router by the *mmap* mechanism [9], so that the user mode virtual router can use its own virtual memory address to directly access packets in the pool. A packet pool is organized as an array of equal size *slots*. Each slot is identified by its index in the array. The size of a slot should be large enough to store one entire packet, i.e., it should be larger than the NIC's MTU. Given $Addr_{pool}$, the virtual address of the first byte in the packet pool, and S_{slot}, the size of a slot, a virtual router can access the ith packet (its index is i) in the pool at address $Addr_{pool} + i \times S_{slot}$.

Asynchronous accessing packets: Each slot in the packet pool has a flag to indicate its "state". A user mode virtual router polls the state flag of a slot (which slot to poll will be discussed in section 3.3 when we present the more detailed design of a virtual router) and only when the state of a slot is "FILLED", which means this slot is filled with a packet, the virtual router can process the packet in that slot. After a virtual router processes a packet in a slot, it changes the state flag of that slot to be "PROCESSED". The EuropaKM monitors the packet pool of the virtual router. If the state flag of a slot is "PROCESSED", EuropaKM sends the packet out to NIC and changes the state flag of that slot to be "EMPTY", which means this slot can be used to store another packet. We see that the state flag of a slot serves as the mutex to coordinate the user mode virtual router and the EuropaKM, and prevent them from concurrently accessing a slot. Hence, changing the state flag should be atomic.

3.3 Virtual Routers

We use OpenVZ [10], an OS-level virtualization scheme, to slice a server into virtual routers. The data plane of a virtual router is implemented by user mode Click. Figure 2 shows the diagram of an Europa virtual router. Each Europa virtual router has its own virtual NICs. A virtual NIC is essentially two ring buffers, denoted as *rxRing* and *txRing*, respectively. Both rxRing and txRing of a virtual NIC store only the indexes of packets in the packet pool. The rxRing has the indexes of packets received from a virtual NIC; the txRing stores the indexes of packets to be sent out via that virtual NIC. Like the packet pool, the rxRing and txRing of all virtual NICs in a virtual router are also shared between EuropaKM and the virtual router via mmap, so that both the virtual router and the EuropaKM can directly read and write them.

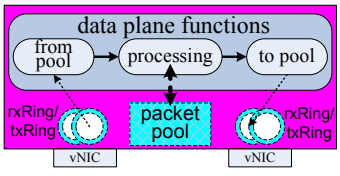


Figure 2: Diagram of the user mode virtual router.

From a virtual router's perspective, receiving a packet from a virtual NIC is essentially reading an index i from the rxRing and accessing the packet at address $Addr_{pool} + i \times S_{slot}$ in the packet pool, which is the "from pool" function in Figure 2; sending a packet via a virtual NIC, which is the "to pool" function in Figure 2, is essentially writ-

ing a packet index into the txRing of the virtual NIC and changing the state of packet slot to be "PROCESSED", so that EuropaKM knows a packet is ready to be sent out. How a virtual router processes packets is up to each virtual router. That is, the "processing" function in Figure 2 is free to be customized by each virtual router.

3.4 EuropaKM

Kernel mode Click is used to implement EuropaKM, whose basic functions are depicted in Figure 3. After receiving a packet from an NIC, EuropaKM first classifies the packet and decides which virtual router it belongs to (the "classify" function in Figure 3). Then the "to pool" function of EuropaKM finds a slot, whose state is "EMPTY", in the virtual router's packet pool and copies that packet into the empty slot. The "to pool" function also writes the index of that slot into rxRing of the virtual NIC and changes the slot state to be 'FILLED". The "from pool" function monitors the txRings to check whether a virtual router has a processed packet. Once the "from pool" function finds that the state of a slot is "PROCESSED", the index of that slot is passed to the "to device" function, which sends out the packet in that slot and changes the slot state to "EMPTY", so it is ready to be loaded with another packet.

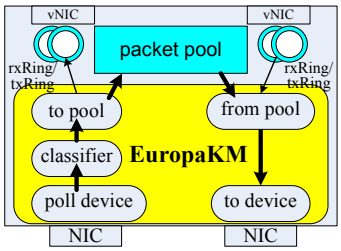


Figure 3: Diagram of EuropaKM.

3.5 Discussion

Security and Isolation: Although Europa shares memory spaces, i.e., packet pools, rxRings, and txRings, between virtual routers and OS kernel, one virtual router can access only its own shared memory spaces. A virtual router doing something wrong, such as writing a corrupted packet in its packet pool, only affects the virtual router itself. We can also implement some sanity checking mechanisms in EuropaKM to enhance its security and avoid jeopardizing the stability of an Europa server. For example, before the EuropaKM accesses a packet in a packet pool, it first checks whether the index of that packet is legal, i.e., the index should be smaller than the number of slots in that packet pool.

Overhead of polling packet state: Both the virtual router and the EuropaKM adopt polling to check the state of a packet. Polling inevitably introduces the CPU usage overhead. Even there is no packet to process, a virtual router still polls for packets and consumes CPU cycles. We believe this CPU overhead should not be an issue for a network virtualization platform. When certain amount of CPU resource is allocated to a virtual router, the virtual router should be entitled to use all its resource. The server virtualization mechanism ensures that a virtual machine does not exceed its CPU resource quota. Hence, even a virtual router uses all its CPU resource, it should not affect other virtual routers hosted in the same Europa server. Besides, polling as fast as possible ensures the next packet to be promptly processed. One possible compromise to reduce the polling overhead is dynamically changing the polling frequency according to the packet incoming rate.

Overhead of packet classification: A packet must be classified after being transferred to main memory via DMA. Then EuropaKM moves it to the packet pool of a virtual router. There is one extra copying operation for each packet. Using virtual queues and classifying packets in NIC hardware can avoid extra packet copying [11]. However, NICs with virtual queues are significantly more expensive. More importantly, those NICs can only classify packets according to MAC address or VLAN tag [11], but other fields in a packet head can be used to indicate which virtual router owns a packet [1,2,12]. Although the scheme used in Europa has one more copying operation for each packet, we believe it is a better tradeoff among performance, flexibility, and cost in building network virtualization substrate at this time.

4 Experimental Evaluation

This section evaluates the data plane performance of Europa. Our experiments show that the packet forwarding speed of Europa is much better than conventional user mode packet forwarding schemes and can match the best known forwarding speed of software router running in commodity hardware.

4.1 Experiment Setting

Figure 4 shows the testbed used in our experiments. The middle machine in Figure 4 runs software routers to forward packet between the sender and receiver machines. All machines are identical commodity desktop PCs. Each one has a 2.66GHz Intel Core2Duo CPU, 4G memory, and two Intel PRO/1000 Gbit NICs. The software router machine runs a customized 2.6.18 Linux kernel. We first apply the OpenVZ patch to a vanilla Linux 2.6.18 kernel and then manually change the source code to include the Click kernel patch. Hence, our kernel supports both OpenVZ and kernel mode Click.

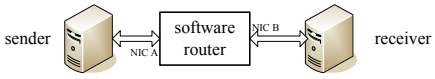


Figure 4: Experiment testbed.

When testing the performance of Europa, we create one or multiple virtual routers in an Europa server. Each virtual router has two virtual NICs, mapped to the two physical NICs of the server, and the Europa virtual router

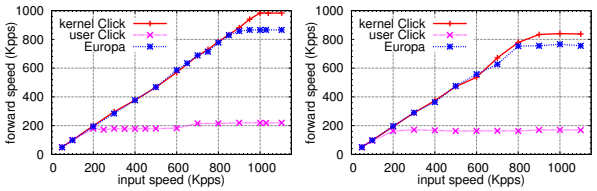
forwards packets between those two virtual NICs. The default packet pool size of each Europa virtual router is 128 packets. We compare the performance of Europa virtual router with two other software routers, i.e., kernel mode Click and user mode Click software routers. Kernel mode Click provides a baseline of the best known packet forwarding speed of software routers running in commodity hardware. User mode Click, on the other hand, presents the forwarding performance of conventional software routers running in user mode, which can be safely customized.

4.2 UDP Experiments

We first use UDP traffic to test the forwarding speed of virtual routers hosted in Europa. The packet forwarding speed is measured in terms of *packets per second* (pps). Minimal length packets (64-byte) are used to stress the virtual routers.

4.2.1 Single virtual router

To understand the raw packet forwarding speed of Europa, we configure the Europa server to host only one virtual router, which is loaded with an IP router configuration with only two entries in its forwarding table. One is to the sender and the other is to the receiver. We also use similar configuration to test the forwarding speed of kernel mode Click and user mode Click running in the same machine. Figure 5(a) plots the forwarding speed results in our experiments. We see that as the packet input speed increases, user mode Click quick reaches a saturation forwarding speed of about 200 Kpps. Kernel mode Click achieves close to 1000 Kpps peak forwarding speed. Europa virtual router can forward packets at about 820 Kpps, which is more than four times the speed of user mode Click router.



(a) small forwarding table (b) large forwarding table

Figure 5: UDP packet forwarding speed.

Next, we test Europa in a more realistic configuration. We extract about $170K$ IP prefixes from a RouteViews [13] BGP table and install them in the forwarding table of an Europa virtual router. The sender machine generates 64-byte UDP packets with random class-C destination IP addresses. The Europa virtual router forwards all incoming packets to the receiver machine. Kernel mode Click and user mode Click routers are also evaluated in similar setting with the large forwarding table. The forwarding speed results are shown in Figure 5(b). We see that Europa still matches the speed of kernel mode Click and is much faster than user mode Click. Figure 5(b) also shows that the forwarding speed gap between kernel mode Click and Europa is smaller than that shown in Figure 5(a). As more CPU cycles are consumed by computational tasks such as IP address lookup, the advantage of kernel mode Click becomes less noticeable, because running those computational tasks in kernel space or user space does not make too much difference in terms of CPU cycle consumption.

4.2.2 Multiple virtual routers

We evaluate the scalability of Europa in terms of hosting multiple virtual routers in one server. The number of concurrent Europa virtual routers is varied from 1 to 10 when measuring the speed of forwarding 64-byte UDP packets. Because section 4.2.1 shows that the performance trends of Europa, kernel Click, and user Click are similar in both small forwarding table and large forwarding table configurations, here we present only the small forwarding table experiment results.

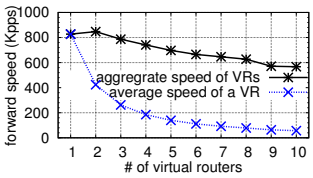


Figure 6: UDP forwarding speed vs. # of virtual routers.

Figure 6 plots the average forwarding speed of each virtual router and the aggregate forwarding speed of all virtual routers hosted in an Europa server. We see that the forwarding speed of a virtual router is inversely proportional to the number of current virtual routers hosted in the Europa server, because multiple virtual routers are competing for CPU and bandwidth resources. As there are more concurrent virtual routers, the aggregate forwarding speed becomes smaller. The reason is that the CPU needs to more frequently switch between different virtual routers to run their data plane processes when there are more virtual routers. The CPU context switching overhead lowers the aggregate forwarding speed of multiple Europa virtual routers. However, we expect that the context switching overhead can be alleviated with the increasing popularity of CPUs with more cores.

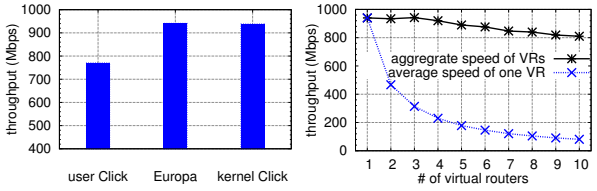
4.3 TCP Experiments

Next, we evaluate the TCP performance of Europa. The *iperf* tool is used to generate TCP traffic between the sender and receiver machines in Figure 4. We do not change any TCP-related parameters of *iperf* but use the default values. Again, only the experiment results of small forwarding table are presented here.

4.3.1 Single virtual router

We test the TCP throughput of a single Europa virtual router, kernel mode Click, and user mode Click. Figure 7(a) plots the experiment results. Europa virtual router achieves almost the same TCP throughput as kernel mode Click. Compared with user mode Click, the throughput

of Europa virtual router is about 22% higher. Because TCP always tries to use large packets, the number of packets forwarded per second is small even the throughput is small to one Gbps line speed. Hence, the advantage of Europa virtual router as compared with user mode Click is not as significant as the UDP experiment results shown in section 4.2. However, we can expect that if faster NICs are used in our experiments, e.g., 10 Gbps NICs, Europa virtual router will show more significant advantage as compared with user mode Click.

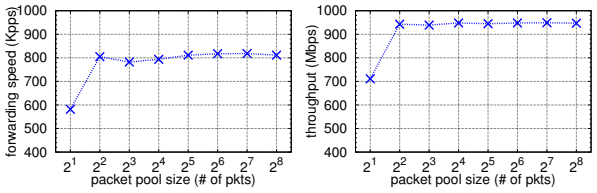


(a) one virtual router (b) multiple virtual routers

Figure 7: TCP throughput.

4.3.2 Multiple virtual routers

We also evaluate the TCP throughput when multiple concurrent virtual routers are hosted in one Europa server. Figure 7(b) shows the average throughput of one virtual router and the aggregate throughput of all virtual routers, when the number of concurrent virtual routers varies from 1 to 10. Not unexpectedly, the average TCP throughput of each Europa virtual router shows inversely proportional property to the number of virtual routers; and the aggregate TCP throughput lowers as more virtual routers are hosted in an Europa server. However, because TCP uses large packets, the aggregate throughput gets about 13% lower only as the number of virtual routers increases from 1 to 10.



(a) UDP (b) TCP

Figure 8: UDP/TCP performance vs. packet pool size.

4.4 Forwarding Performance and Packet Pool Size

Europa uses *packet pool* to share packets between EuropaKM and a user mode virtual router. The packet pool implicitly works as a buffer to cache packets. To study how the size of packet pool affects the forwarding performance of Europa, we run two virtual routers in one Europa server and change the packet pool size of these virtual routers from 2 slots to 256 slots. We measure aggregate 64-byte UDP packets forwarding speed and aggregate TCP throughput for each packet pool size. Figure 8 presents the experiment results. We see that for both UDP and TCP experiments, the forwarding performance of Europa shows little sensitivity to packet pool size larger than 4, because the input traffic in our experiments is close to constant rate.

5 Conclusion

This paper presents EUROPA, a customizable and high-speed network virtualization platform. An EUROPA virtual router runs its data plane in isolated user mode virtual environment. Hence, the data plane of an EUROPA virtual router can be safely customized. In addition to the flexibility of customizing, EUROPA also provides good packet forwarding performance to virtual routers hosted in the platform by adopting an efficient user mode packet forwarding scheme. Unlike existing schemes, Europa uses shared memory to avoid copying packets between user space and kernel and adopts "packet polling" to avoid invoking expensive systems calls. Experiment results show that Europa can closely match the best known software router forwarding speed for both TCP and UDP traffic.

Acknowledgments

This work is partially supported by NSF grants CNS-066618 and CNS-0626617. Dong Yin was a visiting student at UMass, supported by China State Scholarship Fund CSC-2008629080, when this work was performed.

References

[1] A. Bavier and et al, "In VINI veritas: realistic and controlled network experimentation," in *Proceedings of SIGCOMM '06*, 2006.

[2] S. Bhatia and et al., "Trellis: A platform for building flexible, fast virtual networks on commodity hardware," in *Proceedings of ROADS 2008/CoNEXT 2008*, 2008.

[3] P. Druschel and L. L. Peterson, "Fbufs: a high-bandwidth cross-domain transfer facility," in *Proceedings of SOSP '93*, 1993.

[4] "Click Modular Router," http://read.cs.ucla.edu/click/.

[5] K. Argyraki and et al, "Can software routers scale?" in *Proceedings of PRESTO '08*, 2008.

[6] N. Egi and et al., "Towards high performance virtual routers on commodity hardware," in *Proceedings of CONEXT '08*, 2008.

[7] "Intel 64 and IA-32 architectures software developer's manual volume 2B: Instruction set reference, N-Z," Sep. 2009, http://developer.intel.com/design/processor/manuals/253667.pdf.

[8] S. Soltesz and et al, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *EuroSys '07*, Mar. 2007.

[9] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005.

[10] "OpenVZ," http://www.openvz.org/.

[11] S. Chinni and R. Hiremane, "Virtual machine device queues," 2007, Intel white paper, http://software.intel.com/file/1919.

[12] Y. Liao, D. Yin, and L. Gao, "PdP: Parallelizing data plane in virtual network substrate," in *Proceedings of SIGCOMM VISA 2009 workshop*, Aug. 2009.

[13] "Routeviews," http://www.routeviews.org/.