Network Virtualization Substrate with Parallelized Data Plane[☆]

Yong Liao, Dong Yin, Lixin Gao[*]

Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, United States

Abstract

Network virtualization provides the ability to run multiple concurrent virtual networks over a shared substrate. However, it is challenging to design such a platform to host multiple heterogenous and often highly customized virtual networks. Not only high degree of flexibility is desired for virtual networks to customize their functions, fast packet forwarding is also required. This paper presents PdP, a flexible network virtualization platform capable of achieving high speed packet forwarding. A PdP node has multiple machines to perform packet processing for virtual networks hosted in the system. To forward packets in high speed, the data plane of a virtual network in PdP can be allocated with multiple forwarding machines to process packet in parallel. Furthermore, a virtual network in PdP has the freedom to be fully customized. Both the control plane and the data plane of the virtual network run in virtual machines so as to be isolated from other virtual networks. We have built a proof-of-concept prototyping PdP platform using off-the-shelf commodity hardware and open source software. The performance evaluation results show that our system can closely match the best-known packet forwarding speed of software router running in commodity hardware.

Keywords: Network Virtualization, Virtual Network Platform, Parallelization

1. Introduction

Network virtualization provides a powerful way to facilitate testing and deploying network innovations over a shared substrate. Currently the network research community is focusing on building a shared, wide-area experimental platform to support a broad range of research in networking and distributed systems. To that end, and more importantly, toward the long term goal of providing a global infrastructure in which multiple virtual networks, each customized to a specific purpose, could run concurrently, the virtual network substrate must have four key properties: (1) *isolation* between virtual networks to minimize the interference among them; (2) *flexibility* to customize the virtual networks and implement various customized functionalities in the virtual networks; (3) high-speed data plane packet forwarding *performance* to facilitate realistic experiments and attract long term applications; and (4) *low cost* in building that platform to lower the barrier of wide-area deployment.

The challenge of building such a network virtualization substrate is that the four properties, i.e., isolation, flexibility, high performance, and low cost, are often tightly coupled issues in system design, so that we usually have to compromise one in order to improve another one. For example, special purpose hardware can forward packets faster but it usually costs significantly more than commodity hardware. Another dilemma is that in order to achieve better performance, the data plane functions of a virtual network should have direct access to the hardware or run in the privileged domain of the hardware. However, opening low-level and close-to hardware programming interfaces usually results in poor isolation among virtual networks. A buggy function implemented in one virtual network can crash the whole system, e.g., shut down a machine hosting multiple virtual networks. Or a malicious user of

[☆]Part of this work has been published in the Proceedings of VISA'09 - The First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures [1].

[*]Corresponding author. Tel.: +1 (413) 5454548; Fax: +1 (413) 5451993.

Email addresses: yliao@engin.umass.edu (Yong Liao), dyin@engin.umass.edu (Dong Yin), lgao@ecs.umass.edu (Lixin Gao)

the shared platform can easily affect other virtual networks residing at the same substrate. In order to avoid such situation and still offer the desired performance benefits, prior works [2, 3] propose to design and implement a set of well-tested building blocks that can have direct access to the hardware or run in the privileged domain of the hardware. Virtual networks can assemble those building blocks to realize their desired functions. However, that compromises the flexibility because the virtual networks will be limited to the set of provided building blocks.

This paper presents PdP, a new network virtualization platform built from cost-efficient commodity hardware. In designing PdP, we put flexibility as the first priority goal and try to provide each virtual network the freedom of fully customizing both the control plane and the data plane. The basic ideas behind PdP are two-fold. First, both the control plane and the data plane functions of a virtual network hosted in PdP run in virtual machines, so as to provide the isolation among virtual networks and the flexibility to customize each virtual network. Second, there are multiple physical machines serving as the "forwarding engines" in a PdP node. To achieve high speed packet processing, the data plane of a virtual network can have multiple virtual machines (hosted in multiple forwarding engines) running in parallel to perform its data plane packet forwarding functions.

Recently, a few special purpose hardware based network virtualization systems have appeared [4, 5, 3]. Although those systems provide superior packet forwarding performance, they usually are not as flexible as the systems running software routers in the commodity hardware. We can combine PdP with those special purpose hardware-based platforms, in which case PdP complements those systems. In other words, in such a hybrid platform, the PdP subsystem supports highly customized virtual network services; while the special purpose hardware-based subsystem supports virtual network services that can be composed with the set of building blocks provided by the special purpose hardware subsystem.

Although the basic idea behind PdP is promising, implementing this platform is challenging. First, for a PdP node, it is important to ensure that the packet processing performance scales with the number of forwarding engines. The machine coordinating the multiple forwarding engines should not become the bottleneck, especially when the coordination function is implemented by software running in commodity hardware. Second, parallel packet forwarding by multiple forwarding engines can lead to out-of-order packets and thereby result in degraded performance for the up-layer applications, such as those applications using TCP. Therefore, it is important to reduce the amount of out-of-order packets.

In summary, we make three main contributions in this paper. (1) To the best of our knowledge, PdP is the first commodity hardware-based network virtualization platform providing both high degree of customization and viable data processing performance. (2) PdP is the first platform demonstrating the feasibility of scale the packet forwarding speed by parallelizing packet processing in cost efficient commodity hardware. (3) We have built a proof-of-concept PdP node prototype using off-the-shelf commodity hardware and open source software. Our experiment results show that PdP can closely match the best-known packet forwarding speed of software router running in commodity hardware.

The rest of this paper is organized as follows. Section 2 presents related work on network virtualization platforms. Section 3 details the design of PdP. A prototyping implementation of PdP is presented in section 4. Section 5 presents the experiment evaluation results. Section 6 concludes this paper and projects our future work.

2. Related Work

Several network virtualization platforms have been proposed recently. This section provides a brief overview of the existing systems, including both the commodity hardware-based systems and special purpose hardware-based systems.

2.1. Commodity Hardware-based Systems

VINI [6, 7] is a flexible platform that has been deployed in several locations cross the Internet. VINI adopts operating system level virtualization [8] to virtualize a physical server. The virtual routers (virtual machines hosted in one or more physical nodes) are connected by UDP tunnels to build an overlay virtual network. A virtual network hosted in VINI can customize its control plane and data plane with little interference to other virtual networks. However, the packet forwarding in VINI is slow because the virtual network data forwarding function runs in OS user mode.

Trellis [9, 10] also adopts OS level virtualization and the virtual routers are connected by tunnels. A virtual router in Trellis can match the native OS kernel forwarding speed. However, the forwarding performance improvement of Trellis cannot benefit a virtual network that needs to customize its data plane, in which case its packet processing function has to run in OS user mode and that virtual network loses the performance benefit provided by Trellis.

The VRouter project [11] uses Xen [12] to virtualize a physical machine. Xen adopts the so-called paravirtualization [13]. A virtual router can run in either privileged or unprivileged domains of Xen. Running a virtual router in unprivileged domain has unacceptable forwarding performance. Yet, by directly mapping the physical NICs to the virtual routers, an unprivileged domain virtual router can match the native kernel forwarding speed [14]. However, the number of virtual routers would be limited by the number of available NICs in a server. Using NICs with virtual queues and hardware packet classification capability [15] alleviates this problem, but the cost of the NICs would be considerably higher. More importantly, the packet classification in NIC hardware usually classifies packets according to their destination MAC addresses, but wide-area deployable platforms, like VINI and Trellis, use other fields in the packet header to indicate which virtual network a packet belongs to.

The source code merging scheme [2] provides a set of function elements which can run in the privileged domain of underlying hardware. As a result, a virtual network is limited to assemble its data plane using the provided elements.

2.2. Special Purpose Hardware-based Systems

The Supercharging PlanetLab Platform(SPP) [3] separates the control plane and data plane of a virtual network and uses network processor (NP) in virtual network data plane for high speed packet forwarding. SPP opens only the programming interface to control the TCAM hardware in NP. Opening close to hardware programming interface might be a security hole exposed to malicious or reckless users.

There are also platforms built from NetFPGA [16], like those presented in [5, 4]. Although the NetFPGA system facilitates line speed packet forwarding, the platform proposed in [5] cannot support customizing the virtual network data plane function. In the other platform [4], a virtual network can customize its data plane, but the isolation between virtual networks is not good. The NetFPGA board hosting the data planes of multiple virtual networks needs to be shutdown for a short period of time when updating the data plane functions of one virtual network.

Compared with other network virtualization platforms, PdP provides good isolation and flexibility with little packet forwarding performance compromise in commodity hardware. The existing practice and experience of building and deploying network virtualization platforms give us many inspirations in designing PdP. PdP resembles the SPP platform in separating the control plane and the data plane of a virtual network. The deployment experience of VINI and Trellis motivates us to open only the unprivileged domain to virtual networks in PdP. The PC cluster router proposed in [17] inspires us to take advantage of parallization for better data plane packet forwarding performance.

3. PdP: a Network Virtualization Substrate with Parallelized Data Plane

In this section, we present the basic ideas and the detailed design of PdP. When describing PdP, we also point out possible alternative design options in building certain components of PdP and discuss their pros and cons.

3.1. Basic Ideas

The primary design goal of PdP is to provide maximum flexibility and isolation to virtual networks with minimal compromise in packet processing performance. For a virtual router hosted in a PdP node, both its control plane and data plane run in virtual machines (which are also often called guest machines in this paper). The virtualization mechanism that slices a host machine into one or more guest machines, provides the necessary isolation among different virtual networks. Running the control plane and the data plane in guest machines has certain overhead. Although this overhead may not be an issue for the control plane of a virtual network, it can significantly degrade its data plane performance, because the packet processing functions run in the unprivileged domain of the underlying hardware. To compensate the performance degradation of running the data plane in guest machine,

a virtual router in PdP can have multiple guest machines, which run in multiple host machines, to perform its packet processing task. With the parallel processing in multiple machines, a virtual network in PdP can achieve better data plane performance than the virtual networks in other platforms with similar degree of isolation and flexibility. In other words, PdP trades cost (having multiple physical machines to perform the data plane tasks of virtual networks) for better flexibility, isolation, and performance. Since PdP is built from cost-efficient commodity hardware and open source software, the cost increasing should not be substantial.

3.2. PdP Node Architecture

A PdP node consists of a cluster of machines. One machine is dedicated to host the control planes of the virtual routers in the PdP node. We call it the *management host* (denoted by MH). There are multiple machines running the data plane functions of the virtual routers in the PdP node. Those data forwarding machines are the *forwarding engines* (FEs). The data forwarding in multiple FEs is coordinated by the *multiplexer/demultiplexer machine* (denoted by MD).

Both the MH and the FEs are sliced into guest machines using OS level virtualization mechanism [8]. The guest machines hosted in the MH and the FE are called *MH guest machines* and *FE guest machines*, respectively. We choose OS level virtualization because it is efficient and provides good isolation among guest machines. For a virtual router hosted in a PdP node, its control plane, such as the routing process, runs inside an MH guest machine; its data plane packet forwarding function runs in the FE guest machines. The number of FE guest machines serving for a virtual router and the amount of processing power allocated to each FE guest machine depend on how much packet processing power that virtual network claims and the available resources in the PdP node.

The MD machine coordinates the FE guest machines of the virtual routers hosted in a PdP node. Generally saying, the MD distributes all incoming packets to FEs, merges the packets processed by the FEs, and sends the processed packets out to other PdP nodes. For example, once receiving a packet, the MD first decides which virtual network that packet belongs to. Then the MD sends that packet to the corresponding FE guest machines for processing, such as address lookup and traffic shaping. After a packet is processed by the FE guest machine, it is returned to the MD. At that time, the packet is tagged with necessary information (e.g., the outgoing interface) for the MD to decide how to dispatch it out.

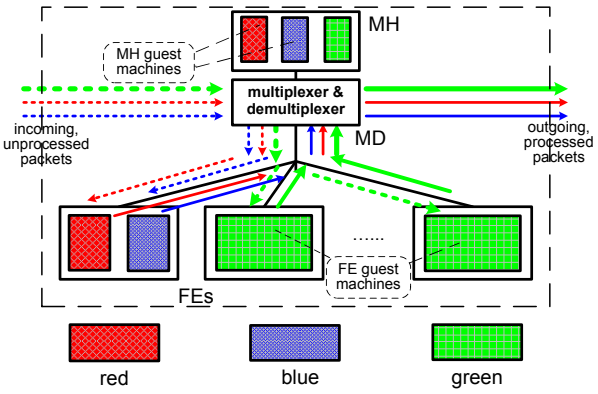


Figure 1: A PdP node example. The dashed arrows represent incoming, unprocessed packets. The solid arrows represent outgoing, processed packets.

We show an example of the PdP node in Figure 1. It hosts three virtual routers serving for three virtual networks, i.e., red, blue, and green. The MH hosts three guest machines and each runs the control plane of a virtual router. Packets belonging to the three virtual networks are classified and distributed from the MD to the FE guest machines. The red and blue virtual networks require little packets processing power, so that one FE is sliced into two FE guest machines, one assigned to the red virtual router and the other assigned to the blue virtual router. The green network requires much more processing power so two FE guest machines, each has all the processing power of one FE, are assigned to the green network. After packets being processed, they are returned to the MD with necessary tags and the MD dispatches those packets out according to those tags.

3.3. Multiplexer and Demultiplexer

The basic functions of the MD machine is to classify incoming packets (from virtual routers in other PdP nodes) to FEs and dispatch processed packets out (to virtual router in other PdP nodes). Those two functions are implemented by the *packet classifier* and the *packet dispatcher*. Figure 2 shows the internal structure of a MD machine.

For each incoming packet, the *packet classifier* first checks whether the packet belongs to a virtual network (e.g., the packet is encapsulated in UDP if

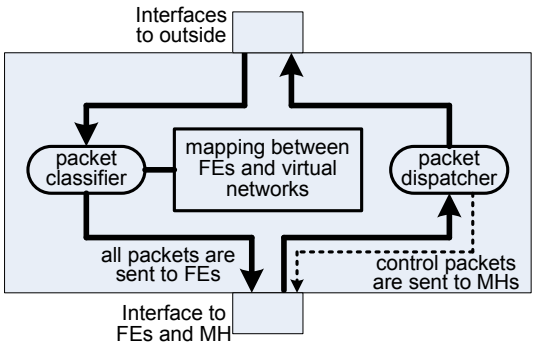


Figure 2: The MD machine.

the virtual links are UDP tunnels between different PdP nodes). If it does, the *packet classifier* further finds out which virtual network that packet belongs to and sends it to the corresponding FE guest machine. The MD has a small table that associates the FE guest machines to the virtual networks. The association between the virtual networks and their FE guest machines should be established in the MD when creating the virtual networks.

After a packet is processed by the FE guest machine and sent back to the MD, it should have been properly tagged with information like which one is the outgoing interface. The tags can be some fields in the header of a lightweight encapsulation mechanism used locally between the MD and the FEs. The *packet dispatcher* checks the tags of the packet and sends the packet out according to the tags.

The MD machine can be a potential bottleneck of a PdP node that constrains it from achieving high packet forwarding speed. We adopt two approaches to speedup the packet processing in the MD machine. First, the *packet classifier* and the *packet dispatcher* functions run in the privileged domain of the MD machine to have minimal overhead in processing packets. Second, the MD machine does not perform any unnecessary "deep" inspections of the packets. Instead, all packets are simply classified to the FEs based on some field in the header (e.g., UDP port). The FEs perform the time-consuming tasks in packet forwarding, such as IP address lookup.

Although we use software to implement the *packet classifier* and the *packet dispatcher* functions by software, those two functions can be implemented by special purpose hardware such as network processor or NetFPGA to achieve better performance. Because the MD machine is not open to virtual networks for customization and programming, using special purpose hardware to implement the MD will not jeopardize the flexibility of PdP.

3.4. Forwarding Engine

The next important component in a PdP node is the FE. In this section we first present the design of the FE. Then we study the resource allocation problem of slicing the FEs and assigning the FE guest machines to virtual routers hosted in a PdP node.

3.4.1. Structure of FE

Figure 3 depicts the structure of an FE. PdP adopt OS level virtualization to slice the FEs. The data plane function of a virtual router hosted in PdP runs in one or more FE guest machines. For an incoming packet, the MD classifies it and sends the packet to an FE guest machine for further processing. The packet processing function running inside the FE guest machine processes the packet, e.g., performing address lookup, encapsulating the packet with proper headers, and labels the processed packet with a set of simple tags. The tags should specify which one is the outgoing interface when the MD dispatches that packet, so that the MD can efficiently send the packet out according to the tags.

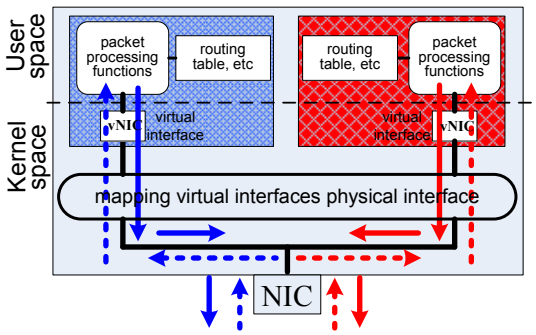


Figure 3: The structure of a Forwarding Engine (FE).

PdP runs the packet processing function inside the FE guest machines, so that the virtualization mechanism provides the isolation among different virtual routers. An alternative design option, which achieves better packet forwarding performance, is letting the packet processing function of a virtual router run in the privileged domain of the FEs, if that virtual router exclusively uses those FEs. However, opening the privileged domain of the FEs could lead to more management overhead because

in some situations the administrator of a PdP system must be involved to resolve the problems. For example, a buggy packet processing function hangs the FEs serving for a virtual network, so that the administrator of the PdP system must hard reboot those machines.

3.4.2. Allocating FEs to Virtual Networks

How to allocate the processing power of the FEs is of importance. The parallel packet processing causes out-of-order packets. Bad FE allocation strategy can exacerbate the situation, which leads to greater impact to the performance of the up-layer protocols such as TCP.

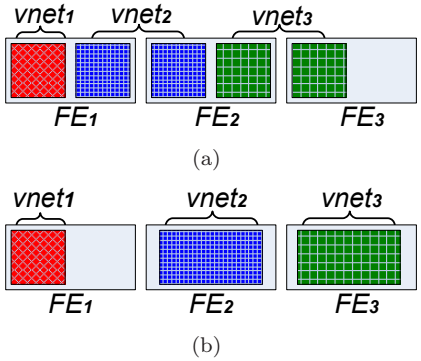


Figure 4: $vnet_1 \sim vnet_3$ get equal processing power in (a) and (b). In (a) there are two FE guest machines, hosted by two FEs, serving for $vnet_2$ and $vnet_3$. In (b) $vnet_2$ (and $vnet_3$) has one FE guest machine, which exclusively uses the processing power of an FE.

Informally, the basic principle should be not slicing the FEs too finely, in order to avoid such a situation that there are a lot of "fragmented" FE guest machines serving for the data plane of a virtual router. For example, suppose there are three FEs ($FE_1 \sim FE_3$) and they are allocated to three virtual networks ($vnet_1 \sim vnet_3$). Slicing and allocating the FEs according to either Figure 4(a) or Figure 4(b) satisfies the processing power requirement of both virtual networks. However, the slicing of FEs as in Figure 4(a) may cause $vnet_2$ and $vnet_3$ to have lots of out-of-order packets. Slicing the FEs as shown in Figure 4(b) is a better choice, because all three virtual networks have their required processing power and they have less out-of-order packets.

In order to minimize the impact of out-of-order packets, we should assign minimal number of FE guest machines to serve for the data plane of a virtual router (the total packet processing power should be enough to satisfy the requirement of the virtual network). Suppose there are n virtual networks ($vnet_1 \sim vnet_n$) and $vnet_i$ requires R_i processing power. Also suppose we have enough FEs in the PdP node and the processing power of each FE is C. If $R_i = kC + r_i$, ($r_i < C$), we should first allocate k FE guest machines to $vnet_i$ and each of them has all the processing power of one FE. Then finding the minimum number of FEs to "pack" the n remainders ($r_1 \sim r_n$) is the classic bin packing problem, which is known to be NP-hard [18]. Considering that new virtual router are created and old ones are removed from a PdP node, we develop an algorithm as shown in Figure 5 to decide the slicing and allocation of FEs in a PdP node. This algorithm adopts a heuristic similar to the "best fit" heuristic used in solving the bin packing problem.

Procedure `SliceAlloc(`R`)`

Input: R, the processing power requirement of a virtual network $vnet$.
Output: The *best fit* slicing and assignment of FEs to satisfy the requirement of $vnet$.

```
1  r=R%C;
2  k=(R − r)/C;
3  for i = 0; i < k; i++ do
4      find an idle FE, create one guest machine with
       C processing power in it, assign that guest
       machine to vnet;
5  FE_* = null;
6  min = BIG_NUM;
7  for FE_i ∈ all FEs and availablePower(FE_i)>r
   do
8      if (availablePower(FE_i) − r) < min then
9          min = availablePower(FE_i) − r;
10         FE_* = FE_i;
11 create a guest machine in FE_* with processing
   power r and assign it to vnet;
```

Figure 5: A *best fit* heuristic algorithm used in allocating the processing power of the FEs.

3.5. Management Host

The MH is sliced into guest machines as well and each guest machine runs the control plane of a virtual router, as depicted in Figure 6.

3.5.1. Handling routing update messages

For a packet containing the control plane message of a virtual router, it is still first sent to one of the FE guest machine of that virtual router by the

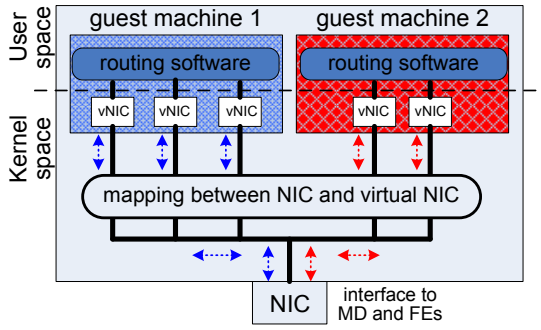


Figure 6: The management host. In this example, the MH hosts two guest machines, which run the control planes of two virtual routers. The arrows represent the routing update messages and control messages of those virtual networks.

MD. The FE guest machine inspects the packet and finds out that the packet's destination is the virtual router itself. Then the FE guest machine labels the packet as control message and sends it back to the MD. According to the tags of that packet, the MD can send the packet to the MH guest machine running the control plane of that virtual router. For a packet sent out by the control plane of a virtual network, the MH should properly encapsulate it and tags that packet with necessary information, like which one is the outgoing interface. Therefore, the MD can send the packet out according to the tags.

3.5.2. Updating the data plane FIB in FE

PdP adopts multicast in updating the data plane FIB of a virtual router, because one virtual router in PdP may have multiple FE guest machines serving as its data plane. Each virtual router hosted in a PdP node is locally assigned a unique multicast address $addr_m$. The FE guest machines of the same virtual router join multicast group $addr_m$. The control plane of a virtual router issues new routes (and control commands) to multicast address $addr_m$, so that the FIB in all its FE guest machines will be updated.

4. Implementation

We have built a prototyping PdP system as shown in Figure 7. All machines are PCs running Linux operating system. For simplicity, we implement the MD and the MH within the same machine. This prototyping PdP node has two external physical interfaces, A and B. We assign two virtual interfaces to each virtual router hosted in this PdP node, one mapped to each physical interface[1], and each virtual router forwards packet between its two interfaces. The *EtherType* field in the MAC header is redefined inside the PdP node as the "tag" to indicate the outgoing interface of a processed packet, or whether a processed packet contains control message.

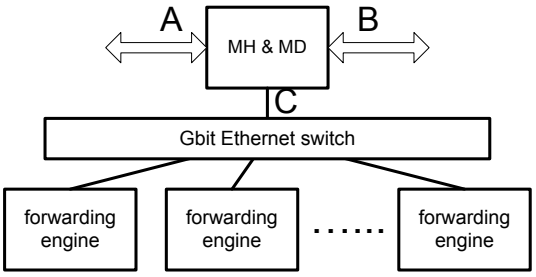


Figure 7: A PdP prototyping system.

The *packet classifier* and *packet dispatcher* functions are implemented using kernel mode Click [19]. The *packet classifier* classifies packets belonging to different virtual networks based on the UDP port numbers (the virtual links in a virtual network are UDP tunnels). The packet processing function, which runs in the FE guest machines, processes each packet, encapsulates the packet with proper headers, labels the packet with a tag to indicate the outgoing interface, and sends it back to the MD. According to the tag labeled to the packet, the *packet dispatcher* in the MD sends that packet out via either interface A or interface B, or sends to the MH guest machine running the virtual router control plane.

We use OpenVZ [20] to slice the MH and the FEs. OpenVZ is an OS level virtualization scheme used in several network virtualization platforms [4, 5, 21, 1]. The packet processing function of each virtual network is implemented by user mode Click running in the FE guest machines. The control plane of each virtual network runs the XORP routing protocol suite [22, 23].

The original user mode Click software router implemented a control interface by which one can connect to the software router via telnet and issue commands to control the router, such as changing the FIB. We patched its source code so that Click opens

[1]Note that this is for testing and prototyping purpose. In reality, each virtual router hosted by the PdP node can have any number of virtual interfaces.

a UDP port and binds that port to a multicast address. Hence, the Click control commands issued to that multicast address can be received and processed by Click.

The XORP routing protocol suite has a Forwarding Engine Abstraction (FEA) layer to support multiple kinds of underlying forwarding engines, such as the native operating system kernel forwarding and Click forwarding. In its current stable version (V1.6), however, XORP does not support using Click running in a remote host as the forwarding engine. We patched the XORP source code so that XORP can populate the FIB of multiple user mode Click instances running in remote hosts by issuing Click control command via multicast.

5. Experiment Evaluation

This section evaluates both the data plane and the control plane performance of virtual networks hosted in PdP, using the prototyping PdP node we have implemented. We focus on IP in evaluating PdP. The basic conclusions of our experiments should apply to virtual networks using protocols other than TCP/IP.

5.1. Experiment Setting and Overview

Figure 8 shows the testbed in our experiments. Two Linux PCs are connected by the router machine via Gbit Ethernet links. The router machine is a virtual router hosted in PdP and we can configure this virtual router to use one, two, or three FEs to forward its packets. All machines are workstation-class PCs equipped with $2.4 \sim 3.0$ GHz Pentium 4 CPU, 1G RAM, and Intel Gbit Ethernet NICs. We enable the NAPI mode [24] in the native Linux NIC driver to avoid the *receive livelock* effect [25].

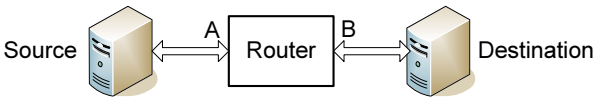


Figure 8: The experiment testbed.

In the following, we first evaluate PdP's data plane performance, including the packet forwarding performance of both UDP and TCP traffic. We also test the the delay for a packet to traverse a PdP node. For the control plane, we first test how fast the control plane of a PdP virtual router can populate its data plane FIB (Forwarding Information Base) in the remote FE guest machines. Then we study the CPU overhead of populating the FIB in FE guest machines and how that affects the packet forwarding performance of PdP virtual router.

5.2. Data Plane Performance

In evaluating the data plane performance of PdP, we compare PdP with user mode Click software router and kernel mode Click software router. The user mode Click provides the baseline performance of network virtualization platforms running software router in the guest machine user mode, such as the VINI platform; the performance of kernel mode Click provides the baseline performance of the best known software router running in commodity hardware.

The purpose of our experiments is to show that the forwarding speed trend of PdP scales with the number of FEs. We can expect that faster speed can be achieved if more powerful hardware, such as server-class PCs, are used in our experiments.

5.2.1. UDP Forwarding Speed

We first use UDP traffic to test the raw packet forwarding speed of PdP. In the testbed shown in Figure 8, the source host runs the udpgen tool shipped with Click to send UDP packets to the destination host. The udpgen tool runs in OS kernel mode and can send out packets at very high speed. The destination host runs the udpcount tool in Click to count the number of received UDP packets. A PdP virtual router forwards the packets between the source and destination hosts.

We had the experiments in which multiple concurrent virtual networks are created in the PdP node. The aggregate forwarding speed of the PdP node, when the number of virtual routers varies from one to three, does not show noticeable difference. Hence, we present only the results where only one virtual router is hosted in the PdP node. If it is not stated explicitly, each FE hosts only one guest machine and the each FE guest machine has all the processing power of the FE.

Forwarding with small FIB: We set the FIB of the virtual router hosted in the PdP node to have only two entries, which point to the source host and the destination host, respectively. The source host sends out 64-byte UDP packets at various rates ranging from $10K$ packets per second (pps) to $400K$

pps. The forwarding speed of the router machine, when it is a PdP virtual router, user mode Click router, or kernel mode Click router, is plotted in Figure 9. The packet loss rate at the router machine is plotted in Figure 10.

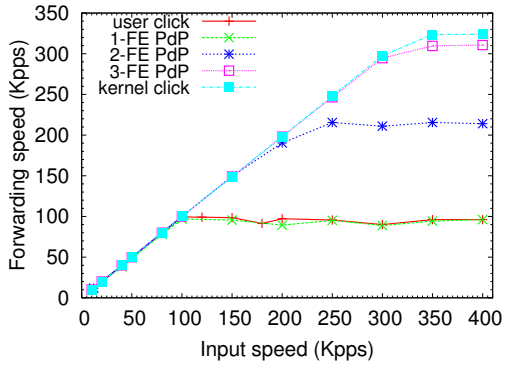


Figure 9: 64-byte UDP packet forwarding speed.

As shown in Figure 9 and Figure 10, when the input speed is lower than certain threshold, the forwarding speed always increases proportionally as input speed increases and the loss rate remains to be close to zero. The peak forwarding speed of PdP virtual router is proportional to the number of FEs and the peak speed of PdP virtual router with three FEs matches the peak speed of kernel mode Click. After the input speed exceeds the threshold, the packet loss rate becomes larger and the forwarding speed becomes stable.

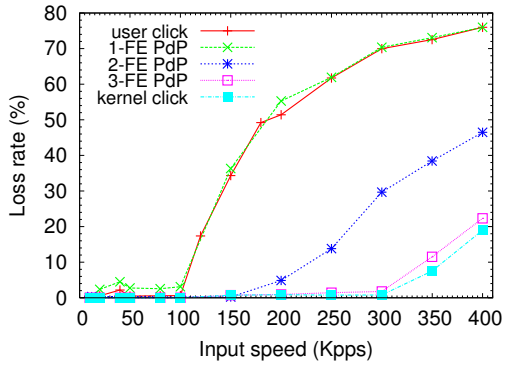


Figure 10: Loss rate in 64-byte UDP packet experiment.

Figure 11 plots the speed in forwarding packets of three typical packet sizes, i.e., 64-byte, 512-byte, and 1500-byte. For the later two packet sizes, we set the packet input speed to saturate the 1 Gbps link. That is, the maximum packet input speed is about $230K$ pps for 512-byte packets and the about $80K$ pps for 1500-byte. As we can see from Figure 11, the forwarding speed becomes lower for larger packets and having more FEs still achieves faster forwarding speed. PdP virtual router with three FEs can match the speed of kernel mode Click in both the 512-byte packet experiment and the 1500-byte packet experiment. Allocating two FE machines instead of one to the PdP virtual router can double its forwarding speed. However, increasing the number of FEs from two to three does not show proportional forwarding speed enhancement because of the bandwidth limit of Gbit Ethernet link.

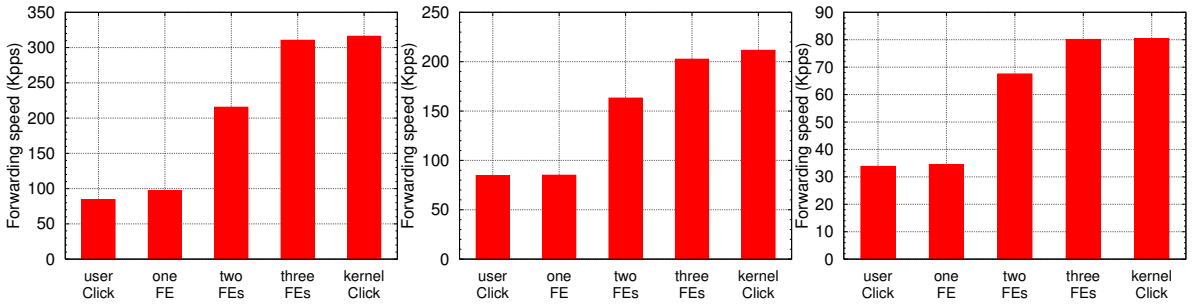
Forwarding with large FIB: When the number of entries in the FIB is small, the IP address lookup time is ignorable due to the "warm cache" effect [26]. To study the forwarding performance of the PdP virtual router in case of large FIB, we configure a table with more than $170K$ entries in the FE guest machines of the virtual router and repeat the above UDP packet experiments. In the router machine, all routes have the same *nexthop*, which is the destination host. To avoid the warm cache effect, the source host sends out UDP packets with randomly selected unicast destination IP addresses. The forwarding speed experiment results, for three typical packet sizes, are plotted in Figure 12.

The results in Figure 12 show that the PdP virtual router still performs better than the user mode Click software router and matches the speed of kernel mode Click. However, the forwarding speed gets lower when using large FIB, especially for 64-byte packet experiment. For large packets, the forwarding speed does not show much degradation because the packet input speed is slow (due to the link bandwidth limit) and the IP address lookup time is not the significant part in packet processing.

5.2.2. Out-of-Order Packets and TCP Throughput

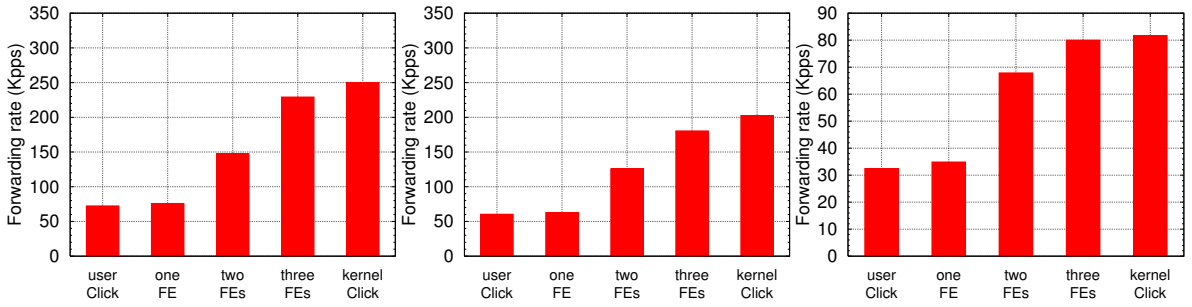
The experiments using UDP traffic test only the raw packet forwarding speed of virtual router hosted in PdP. Most popular network applications use TCP protocol and the actual throughput achieved by TCP depends on more factors such as packet arriving order, round trip time etc.

Out-of-order packets: Packet out-of-order is a challenging problem for any parallel processing-based systems. The following experiment is to quantify how the parallelization of packet processing in the PdP virtual router affects packet out-of-order in TCP. We configure the source host to generate a TCP session using iperf and capture all the packets at the destination host. Then we use the



(a) 64-byte packet (b) 512-byte packet (c) 1500-byte packet

Figure 11: Packet forwarding speed when FIB is small (2 entries).



(a) 64-byte packet (b) 512-byte packet (c) 1500-byte packet

Figure 12: Packet forwarding speed when FIB is large (more than $170K$ entries).

Expert Info tool in wireshark [27] to analyze the out-of-order packets. The percentages of out-of-order packets, when the PdP virtual router has one, two, and three FEs, are shown in Table 1.

	1 FE	**2 FEs**	**3 FEs**
% of out-of-order pkts	0.31%	10.19%	13.02%

Table 1: Percentages of out-of-order packets. MD distributes packets to FE guest machines based on round-robin.

When more than one FE guest machines are used by the PdP virtual router, about $10\% \sim 13\%$ packets in the TCP session are out-of-order and there is no significant difference between the experiments using two and three FE guest machines. Although considerable number of packets in the TCP session are out-of-order packets, as will be shown later, we still get decent TCP throughput.

Next we evaluate how the strategy of the *packet classifier* running in the MD affects the packet out-or-order. We use two identical FEs in the PdP node and each FE hosts one guest machine. We tune the setting of OpenVZ so that one FE guest machines has 75% of the CPU cycles of an FE and the other FE guest machines has 50% of the CPU cycles of an FE. The *packet classifiers* in MD uses two packet distributing strategies. One is sending packets in round-robin manner; the other is sending different number of packets based the allocated CPU cycles of the FE guest machines, i.e., for every 5 packets, sending packets 1, 3, 5 to the FE guest machine with 75% CPU cycles and sending packets 2, 4 to the FE guest machine with 50% CPU cycles. Table 2 shows the results.

	round-robin	**proportional**
% of out-of-order pkts	12.27%	10.02%

Table 2: Out-of-order packets when MD uses round-robin scheduling and proportional scheduling.

The results in Table 2 tell us that less out-of-order packets will occur if the *packet classifier* in MD takes into account the packet processing power of FE guest machines instead of blindly sending packet to FE guest machines in a round-robin manner.

TCP throughput: We configure an iperf server to

run in the destination host. An iperf client runs in the source host and sends TCP traffic to the iperf server. When multiple FEs are used by the PdP virtual router, the MD sends packets to them in a round-robin manner. Both small FIB and large FIB are tested in our experiments. However, the TCP throughput in those two cases does not show significant difference. This is consistent with our UDP packet forwarding speed experiment results where large packets are used, because most of the packets in the TCP sessions are large packets (more than 512 bytes). Figure 13 plots the TCP throughput in our experiments using small FIB.

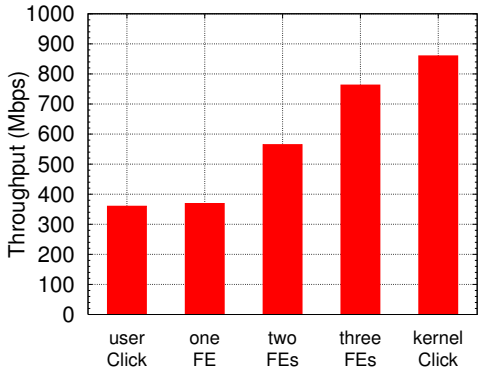


Figure 13: The TCP throughput.

Our experiments show that the PdP virtual router with one FE guest machine achieves similar TCP throughput as user mode Click router. Although there are a lot of out-of-order packets, the PdP virtual router with two or three FE guest machines demonstrates significant improvement in TCP throughput compared with user mode Click and PdP virtual router with one FE guest machine.

5.2.3. Packet Traverse Time

As every packet needs to forwarded between multiple PCs in a PdP virtual router, one can expect the it takes longer for a packet to traverse a PdP node. We use the following experiment to evaluate the extra delay in the packet traverse time. Here the source host in Figure 8 uses ping to send ICMP packets to the destination host. We record average round-trip-time (RTT) reported by ping and show the results in Table 3.

Table 3 shows us that one PdP node adds about $0.17ms$ extra delay to the RTT, compared with the kernel mode Click router. That means a packet needs about $0.085ms$ additional time to traverse a PdP node once. According to the measurement

	user Click	**PdP**	**kernel Click**
RTT (ms)	0.208	0.296	0.132

Table 3: Round trip time of traversing a PdP node.

study in [28], most hosts in Internet are about 14 hops away from a university probing site and the average RTT from those hosts to the probing site is about $80ms$. Therefore, if PdP is widely deployed in Internet and each PdP node adds $0.17ms$ additional RTT delay, the total additional RTT delay would be about $2.4ms$, which is negligible considering an $80ms$ average RTT in Internet. As one can see from Table 3, if comparing PdP with user mode Click router, the extra delay in packet traverse time would be even more insignificant.

5.3. Control Plane Overhead and Its Impact to Data Plane Performance

How fast the control plane of a PdP virtual router can update its data plane is an important issue we should consider, because the control plane and data plane of a virtual router hosted in PdP are located at different machines. Control plane updating data plane FIB also introduces certain overhead in the FE guest machines running the virtual router data plane. We also evaluate how that affects the packet forwarding in data plane.

FIB population time: We setup a PdP virtual router with one FE guest machine, which exclusively uses one FE. That virtual router runs the XORP BGP routing process as its control plane. Another machine runs the lbgps tool [29] to inject routes into the XORP BGP routing process of the virtual router. We vary the size of the routing tables injected by lbgps from 10 entries to 200,000 entries. While the control plane XORP is updating the FIB of the data plane, we monitor the Click running in the FE guest machine and record when the first and the last routes are received and installed in Click. For each table size, we repeat the experiment for five times. The average time used by the PdP virtual router control plane to populate its data plane FIB, as a function of the table size, is plotted in Figure 14.

From Figure 14 we can see that the FIB population time increases linearly as the table size grows. When the table has 200,000 entries, which is about half of the prefixes in a full Internet BGP table, it can take about one and half minutes to populate

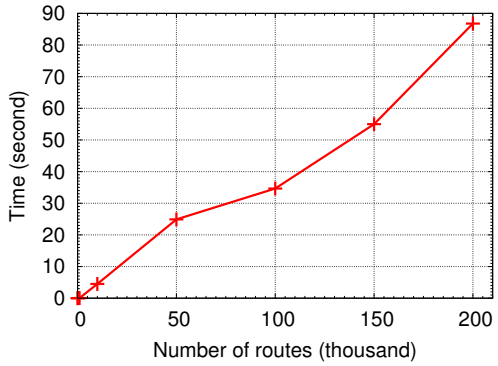


Figure 14: FIB population time.

the data plane FIB of the virtual routers. However, considering the FIBs are usually incrementally updated, we can expect that usually a PdP virtual router can update its data plane FIBs quickly. For example, in our experiments, it takes only 0.4 seconds for the virtual network control plane to populate 1,000 routes into its data plane FIB.

In our experiments, we count the number of routes in the date plane FIB after the control plane finishes the route updating to check whether all routes are installed in the data plane FIB. Although multicast uses UDP, our experiments show that the virtual router data plane always successfully receives all the routes sent by its control plane and installs them in the FIB.

FIB population overhead: In Figure 15, we plot the CPU usage of the FE guest machine when the PdP virtual router control plane is populating its data plane FIB using a table with $200K$ entries. Figure 15 shows that populating the data plane FIB is not very expensive in terms of CPU usage in the FE guest machine. During the FIB populating process, the CPU usage of the FE guest machine never exceeds 25%. On average, the CPU usage of the FE guest machine is about 14%.

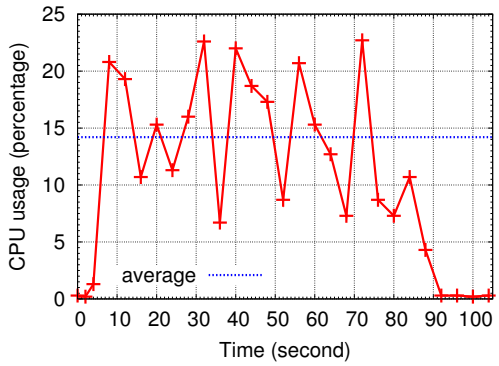


Figure 15: FE guest machine CPU usage during populating 200,000 routes into the FIB .

Impact to data plane: To understand the impact of the control plane overhead to the data plane performance, we repeat the TCP throughput experiment when the virtual router control plane is updating its data plane FIB with a table that has $200K$ entries. The experiment results are plotted in Figure 16. Our experiment shows that the TCP throughput is about 20% slower when the control plane of the PdP virtual router is updating its data plane FIB, which is about proportional to the CPU usage overhead in populating FIB. The TCP throughput in our experiment ranges in $300M \sim 320M$.

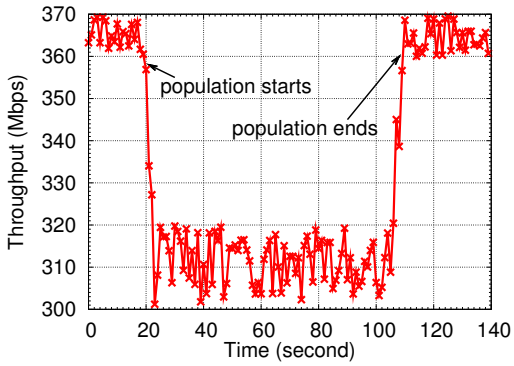


Figure 16: TCP throughput when populating 200,000 routes into the FIB.

We should note that populating a large FIB (like the table used in our experiment with $200K$ entries) into the data plane happens only in some special cases. For example, a multi-homed AS in Internet loses one of its provider links so half of its routes will be changed. In most cases, the data plane FIBs are incrementally updated and only small number of routes will be changed. As Figure 14 shows, updating 1,000 routes takes about 0.4 seconds. Hence, we can expect that in most cases the control plane overhead will not have noticeable to the data plane performance.

6. Conclusion

In this paper we present PdP, a full programmable and high speed network virtualization platform. PdP is built from cost-efficient commodity hardware and open source software. A virtual network hosted in PdP can have complete control over its control plane and data plane without interfering other virtual networks. The key ideas behind PdP are two-fold: running virtual network control plane and data plane in guest machines for better isolation and flexibility; having multiple guest

machines working in parallel to achieve high speed packet processing. We have built a prototype of the PdP node. The performance measurement shows that for both UDP and TCP traffic, a virtual router hosted in PdP can closely match the best-known packet forwarding speed of software routers running in commodity hardware. Our experiment also shows that the control plane of a virtual router hosted in PdP can update its data plane FIB without too much overhead.

7. Acknowledgements

This work is supported by U.S. NSF grants CNS-0626618 and CNS-0831940.

Reference

[1] Y. Liao, D. Yin, L. Gao, PdP: Parallelizing Data Plane in Virtual Network Substrate, in: Proceedings of VISA 2009 - The First ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures, 2009.

[2] E. Keller, E. Green, Virtualizing the data plane through source code merging, in: PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, 2008.

[3] J. S. Turner,, P. Crowley,, J. DeHart,, A. Freestone,, B. Heller,, F. Kuhns,, S. Kumar,, J. Lockwood,, J. Lu,, M. Wilson,, C. Wiseman,, D. Zar,, Supercharging planetlab: a high performance, multi-application, overlay network platform, in: SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, 2007.

[4] D. Unnikrishnan, R. Vadlamani, Y. Liao, A. Dwaraki, J. Crenne, L. Gao, R. Tessier, Scalable network virtualization using FPGAs, in: Proceedings of FPGA 2010: Eighteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2010.

[5] M. B. Anwer, N. Feamster, Building a fast, virtualized data plane with programmable hardware, in: Proceedings of VISA 2009 - The First ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures, 2009.

[6] A. Bavier, N. Feamster, M. Huang, L. Peterson, J. Rexford, In VINI veritas: realistic and controlled network experimentation, in: SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, 2006.

[7] VINI: A virtual network infrastructure, `http://www.vini-veritas.net`.

[8] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, L. Peterson, in: Proceedings of EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, ACM, New York, NY, USA, 2007, pp. 275–287.

[9] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, J. Rexford, Hosting virtual networks on commodity hardware, Tech. rep., Princeton University (Nov. 2007).

[10] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, J. Rexford, Trellis: A platform for building flexible, fast virtual networks on commodity hardware, in: Proceedings of ROAD' 08: 3rd International Workshop on Real Overlays and Distributed Systems, 2008.

[11] The Virtual Router Project, `http://nrg.cs.ucl.ac.uk/vrouter`.

[12] The Xen hypervisor, `http://www.xen.org`.

[13] VMWare, Understanding full virtualization, paravirtualization, and hardware assist, vMWare White Papre: `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf` (Sep. 2007).

[14] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, L. Mathy, Towards high performance virtual routers on commodity hardware, in: CoNEXT '08: Proceedings of the 4th ACM International Conference on emerging Networking EXperiments and Technologies, 2008.

[15] S. Chinni, R. Hiremane, Virtual machine device queues, Intel white paper, `http://software.intel.com/file/1919` (2007).

[16] J. Naous, G. Gibb, S. Bolouki, N. McKeown, NetFPGA: Reusable router architecture for experimental research, in: Proceedings of PRESTO '08, 2008.

[17] K. Argyraki, S. Baset, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, S. Nedevschi, S. Ratnasamy, Can software routers scale?, in: PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, 2008.

[18] P. Crescenzi, V. Kann, A compendium of NP optimization problems, online document: `http://www.nada.kth.se/~viggo/problemlist/`.

[19] The Click Modular Router, `http://read.cs.ucla.edu/click/`.

[20] OpenVZ: container-based virtualization for Linux, `http://www.openvz.org`.

[21] Y. Wang,, E. Keller,, B. Biskeborn,, J. van der Merwe,, J. Rexford,, Virtual routers on the move: live router migration as a network-management primitive, SIGCOMM Comput. Commun. Rev. 38 (4) (2008) 231–242. doi:http://doi.acm.org/10.1145/1402946.1402985.

[22] M. Handley, O. Hodson, E. Kohler, Xorp: an open platform for network research, SIGCOMM Comput. Commun. Rev. 33 (1) (2003) 53–57. doi:http://doi.acm.org/10.1145/774763.774771.

[23] XORP: eXtensible Open Router Platform, `http://www.xorp.org`.

[24] NAPI: a new device driver packet processing framework, `http://www.linuxfoundation.org/collaborate/workgroups/networking/napi`.

[25] J. C. Mogul, K. K. Ramakrishnan, Eliminating receive livelock in an interrupt-driven kernel, ACM Trans. Comput. Syst. 15 (3) (1997) 217–252.

[26] Click element documentation, `http://read.cs.ucla.edu/click/elements/iproutetable`.

[27] wireshark, `http://www.wireshark.org`.

[28] A. Fei, G. Pei, R. Liu, L. Zhang, Measurements on delay and hop-count of the internet, in: IEEE GLOBECOM'98 - Internet Mini-Conference, 1998.

[29] A Lightweight BGP Speaker, `http://rio.ecs.umass.edu/~yliao/html/software/lbgps-v0.1.tgz`.