Scalable Network Virtualization Using FPGAs

Deepak Unnikrishnan†, Ramakrishna Vadlamani†, Yong Liao†, Abhishek Dwaraki†, Jérémie Crenne*,
Lixin Gao†, and Russell Tessier†
{unnikrishnan,vadlamani,yliao,adwaraki,lgao,tessier}@ecs.umass.edu, jeremie.crenne@univ-ubs.fr

†Electrical and Computer Engineering
University of Massachusetts, Amherst
MA 01003

*European University of Brittany
UBS – Lab - STICC
Lorient, France 56100

ABSTRACT

Recent virtual network implementations have shown the capability to implement multiple network data planes using a shared hardware substrate. In this project, a new scalable virtual networking data plane is demonstrated which combines the performance efficiency of FPGA hardware with the flexibility of software running on a commodity PC. Multiple virtual router data planes are implemented using a Virtex II-based NetFPGA card to accommodate virtual networks requiring superior packet forwarding performance. Numerous additional data planes for virtual networks which require less bandwidth and slower forwarding speeds are implemented on a commodity PC server via software routers. Through experimentation, we determine that a throughput improvement of up to two orders of magnitude can be achieved for FPGA-based virtual routers versus a software-based virtual router implementation. Dynamic FPGA reconfiguration is supported to adapt to changing networking needs. System scalability is demonstrated for up to 15 virtual routers.

Categories and Subject Descriptors

C.2 [**Computer communication networks**]: Internetworking routers

General Terms

Design

Keywords

Virtual networks, FPGA

FPGA'10, February 21–23, 2010, Monterey, California, USA.
Copyright 2010 ACM 978-1-60558-911-4/10/02...$10.00.

1. INTRODUCTION

As the Internet evolves, increasingly diverse network applications will be deployed to accommodate business and social needs. Often, network applications call for strikingly divergent performance requirements in terms of security, predictability, and throughput. Although physically separate networks could be constructed to meet these varied service requirements, a common physical substrate minimizes equipment investment, operating cost, and power consumption. Network virtualization, which supports the simultaneous operation of multiple virtual networks over shared network resources, provides a powerful way to customize each network to a specific purpose and service requirement while minimizing hardware resources [8]. Although virtual networks use the same hardware resources, individual network data and control planes are effectively isolated, providing protection against misconfigurations and intrusions from other networks. Such an approach also allows for customized protocols and/or configurations that may be experimental in nature without burdening pre-existing networks.

The introduction of network virtualization has recently led to a collection of mostly software-based virtualization solutions. Although these implementations exhibit substantial flexibility and support for tens of simultaneous networks [2][5], the serial nature of general purpose microprocessors limits their achievable performance. This constraint points towards a configurable approach with significant fine-grained parallelism and specialization, i.e., the implementation of virtual networks in FPGA hardware to achieve both configurability and performance goals. Although improved performance versus software-based approaches has been demonstrated in existing FPGA-based systems [1][22], the scalability of these techniques to tens of networks is limited by the logic and memory resources of the target FPGA. More importantly, these previous systems do not gracefully support virtual network reconfiguration or re-programming, which are much desired features in a network virtualization substrate.

Our scalable virtual networking approach overcomes the limitations of previous approaches by seamlessly integrating FPGA-based virtual routers with software-based virtual routers. The highest-throughput routers are mapped to hardware while remaining virtual routers are implemented by software routers running in virtual machines hosted in a commodity PC. Both hardware and software routers support the same user-defined protocols and isolation techniques in an approach which is transparent to end users. A virtual router in our system can be migrated between the FPGA and the PC server. Decisions regarding which virtual routers should be hosted in the FPGA hardware can be made dynamically as user needs change. Dynamic reconfiguration is used to update FPGA virtual routers as networking needs change.

To determine the real-time performance of the new virtual networking environment, a parameterizable virtual router design has been implemented in a Virtex II-based NetFPGA board [12]. The board is included in a Linux-based PC which is sliced into virtual machines using the OpenVZ virtualization scheme [23]. Our current FPGA architecture can simultaneously support up to five different virtual networks. The data planes of virtual networks requiring superior packet forwarding speed are hosted in FPGA hardware. A performance evaluation shows that when multiple virtual routers are hosted in FPGA hardware, each of them can achieve close to line speed packet forwarding. The virtual routers running in the FPGA can be migrated to software routers running in the OpenVZ containers when the FPGA needs to be reconfigured.

The rest of the paper is organized as follows. Section 2 introduces virtual networks and current software and hardware-based virtualization techniques. Section 3 presents the architecture of our virtual routing system and compares its structure with previous hardware- and software-based implementations. The experimental approach used for virtual network evaluation is described in Section 4. In Section 5, the throughput, latency, and scalability results of our work are detailed. Section 6 summarizes the paper and offers directions for future work.

2. BACKGROUND

2.1 Network Virtualization

Network virtualization has been proposed as a powerful approach to facilitate the testing and implementation of network innovations using a shared substrate [2][18]. In a network virtualization infrastructure, concurrent virtual networks are implemented in shared networking hardware. Such an approach reduces the expense of hardware and provides a platform for researchers to independently deploy and evaluate innovative networking techniques. To those who create the virtual networks, each independent virtual network represents an isolated and customizable platform with prespecified routing protocols, bandwidth guarantees, and quality of service (QoS). Figure 1 shows a network infrastructure, which is virtualized to support a red virtual network and a blue virtual network. Each virtual network might run different routing processes (with the same or different routing protocols) and therefore might have different views of the network topology. For example, Figure 2 shows a topology which differs from the one in Figure 3 although both virtual networks run on the same physical network.

Figure 4 shows the structure of a physical router in a virtualization substrate, which is partitioned into multiple virtual routers. A virtual router consists of two major parts: a control plane, where the routing processes exchange and maintain routing information; and a data plane, where the forwarding information base (FIB) stores the forwarding route entries and performs packet forwarding.

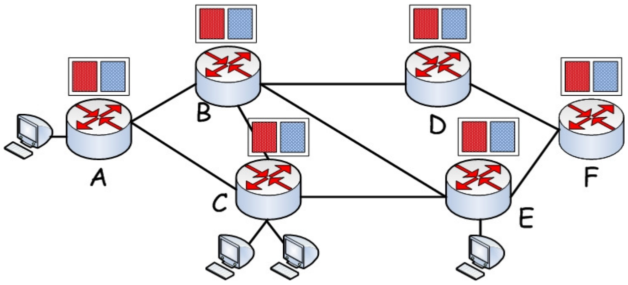

Figure 1. Virtual networks

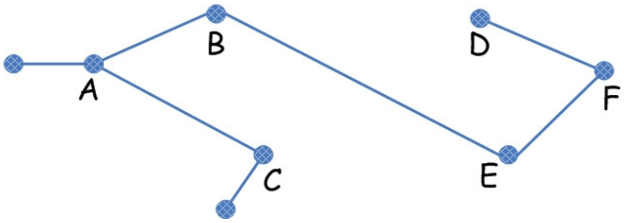

Figure 2. Blue virtual networks

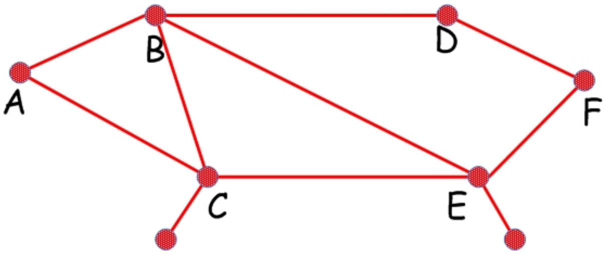

Figure 3. Red virtual networks

The virtual routers are independent of each other and can run different routing, addressing and forwarding schemes. A physical link is multiplexed into virtual links, which connect the relevant virtual routers into virtual networks. Any virtual router joining the virtual network is marked with this color (e.g. red or blue) and data packets are colored in a similar fashion. The physical router provides DEMUX and MUX circuitry for the hosted virtual routers. After exiting a physical link, a colored packet will be delivered by the DEMUX to the virtual router with the same color. When packets are emitted from a virtual router, it is colored with the router's color at the MUX before it enters the physical link. Because of this packet-level separation, a virtual router can only communicate with virtual routers of the same color. As a result, a network infrastructure can run multiple virtual networks in parallel and each virtual network can run any addressing, routing and forwarding scheme without interfering with a different virtual network.

Flexibility is a key requirement of any virtual networking substrate. Specifically, a virtual network must offer maximum control over its data and control planes. For example, the deployment of new addressing schemes such as ROFL [16] requires customization of nearly all aspects of the network core such as the routing protocol and the address lookup algorithm. Other examples include QoS schemes that require certain queuing and scheduling approaches and security mechanisms such as network anonymity or onion routing [19][20]. To attract existing applications whose performance requirements may scale over time, superior data plane performance is also of utmost importance.

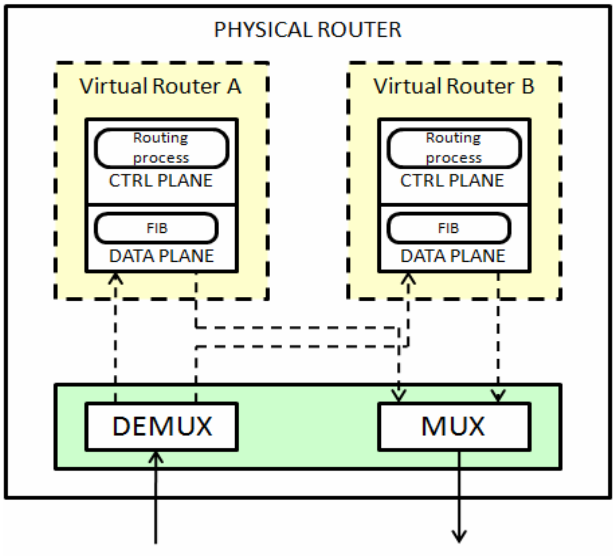

Figure 4. Virtual router architecture

The customization of existing proprietary network devices to support virtual networking is challenging, if not impossible. Contemporary network devices typically do not provide the interfaces necessary to enable programmability. Most existing network systems employ application specific integrated circuits (ASICs). Although ASICs typically achieve high performance, they do not have the flexibility needed for service customization. For example, the Supercharging PlanetLab platform [17] only provides a customizable forwarding table interface, which makes it hard to support innovations such as network anonymity, onion routing and QoS schemes.

2.2 Network Virtualization with Software

To allow for the rapid allocation of system resources and a flexible programming environment, several network virtualization systems have recently been implemented using off-the-shelf hardware with existing host virtualization technology [4][11]. The execution of customized software on general-purpose microprocessors provides the flexibility and programmability needed to build customizable virtual networks.

Bhatia, et al [5] developed a virtual networking environment which can be scaled to 60 independent networks. This system allows for individual network customization and the use of a commodity operating system which can support a variety of services, including tunneling. Packet forwarding is performed in the kernel under application control. Keller and Green [21] proposed a system which allows for customized packet handling for each data plane in a virtualized network. This system uses an unvirtualized Linux kernel to host multiple concurrent data planes implemented in Click [29]. Packet handling is specified as an interconnected graph of networking functions.

Although the substantial progress of these and other virtual networking systems is important [4][7], the serial nature of general-purpose microprocessors limits the achievable performance of software-based virtual network devices. Software-based data plane implementations can exhibit statistical variations in observed network parameters due to jitter and resource contention. For example, in container-based virtualization and full virtualization techniques [2], each virtual network resource must contend for hardware and operating system resources such as CPU cycles, bandwidth and physical memory. Although the extent of contention could be reduced through CPU/memory reservations and real time priority assignments [2] to each virtual network, these approaches tend to degrade flexibility in experiments.

2.3 Network Virtualization with FPGAs

Over the past eight years, FPGAs have been used extensively to evaluate and implement network routers. These programmable components are now viewed as prime candidates for next generation network router implementations [13]. Initial work in FPGA-implemented routers focused on basic router implementations [10], network packet intrusion detection [9][15], and buffer sizing [3]. Two very recent efforts assess the use of FPGAs in virtual networking.

In Anwer, et al [1], a NetFPGA board is used to implement up to eight identical virtual routing data planes in a single Virtex II Pro. The control planes are implemented in software with the help of virtual containers in the host operating system. Physical links are virtualized by associating each physical NetFPGA network port with one or more virtual ports. Although this setup is shown to provide twice as much throughput as a software kernel router, a number of limitations exist. All data planes are identical; only routing table differentiation is supported. Additionally, the resources of the FPGA provide a hard cap on the number of supported virtual networks. Finally, no support for dynamic reconfiguration of the FPGA to support changing networking needs is mentioned.

Multiple different virtual data planes have recently been implemented [22] using a NetFPGA platform. Up to two data planes can be specified by designers and implemented in FPGA hardware. A series of configuration registers are available which allow for real-time updates to virtual routing protocols. No discussion of scalability, logic reconfiguration, or data plane customization is provided.

3. SCALABLE VIRTUAL NETWORKING

Our new network virtualization system makes two specific extensions to previously implemented systems:

1. The number of virtual networks supported in our system is not limited to the amount of FPGA logic. Multiple high-throughput, customized data planes are implemented in hardware while a scalable number of slower additional data planes are simultaneously implemented in software. Although the number of high performance virtual data planes is limited by the available FPGA logic, users have

the additional flexibility of implementing slower data planes in host software.

2. The FPGA implementation of the virtual data planes is not static. The FPGA can be dynamically reconfigured over time while the network is operational to support alternate planes.

In the following subsections, an overview of our implementation and run-time environment is detailed.

3.1 System Overview

The high-level architecture of our system is shown in Figure 5. In this setup, virtual routers that require highest throughout and lowest latency are implemented on a Virtex II-Pro 50 on the NetFPGA platform [14] while additional software virtual routers are implemented in OpenVZ [23] containers running in the PC. The NetFPGA platform consists of four 1 Gbps Ethernet interfaces, a 33 MHz PCI interface, 64 MB of DDR2 DRAM and two 32 MB SRAMs. The hardware data path of the NetFPGA platform is implemented as a pipeline of fully customizable modules. The forwarding tables of the hardware virtual routers are implemented in BRAM and SRL16E memories within the FPGA. Forwarding tables can be updated from software through the PCI interface. The PCI interface facilitates flexible control plane implementations in software.

In addition to the NetFPGA board, our system includes a PC server to host the software virtual routers. The PC server is sliced into virtual machines using OpenVZ [23]. OpenVZ is a lightweight virtualization approach used in several network virtualization systems [24][25] and it is included in major Linux distributions. OpenVZ virtualizes a physical server at the operating system level. The OpenVZ kernel allows multiple isolated user-space instances, which are called virtual machines or containers. Each virtual machine performs and executes like a stand-alone server. The OpenVZ kernel provides the resource management mechanisms to allocate resources such as CPU cycles and disk storage space to the virtual machines. Compared with other virtualization approaches, such as full virtualization [26] and paravirtualization [27], the OS-level virtualization provides the best performance and scalability. The performance difference between a virtual machine in OpenVZ and a standalone server is almost negligible [28].

When the number of virtual networks exceeds the available FPGA hardware resource capacity, additional routers are spawned in the host software on the PC server. Since software virtual routers must be effectively isolated from each other, they are hosted in separate OpenVZ containers that guarantee a fair share of CPU cycles and physical memory to each virtual router. Each instance of the OpenVZ container executes a user mode Click modular router [29] to process the packets. The forwarding functions of Click can be customized according to the virtual network creator's preferences.

When a packet arrives at an Ethernet interface (PHY), the destination virtual IP address (DST VIP) in the packet header (Figure 6) is used to determine the location of its virtual router. If the packet is associated with a hardware virtual router, it is processed by the corresponding hardware router. Otherwise, it is transmitted to the host software via the PCI bus. A software bridge provides a mux/demux interface between the PCI bus and multiple OpenVZ routers. Periodically, the virtual networks in the FPGA are reconfigured to take into account changes in the bandwidth demands and routing characteristics. While the FPGA is being reconfigured, all traffic is routed by the host software. When reconfiguration is finished, selected virtual networks are shifted back to the hardware based on their performance requirements. The adoption of network virtualization for realistic network experiments requires widespread deployment across multiple sites. Tunneling transforms data packets into formats that enable them to be transmitted on networks that have incompatible address spaces and protocols. Our system supports the use of layer 3 virtualization based on IPIP tunneling [30]. In this tunneling approach, each node in a virtual network has a virtual IP address assigned from a private address space. To transmit a packet to another virtual node in the private address space, packet data is encapsulated in an IPv4 wrapper, as shown in Figure 6.

The datagram is subsequently tunneled through routers to the next virtual node. When the packet reaches a virtual router, the inner virtual IP address is used to identify the next virtual hop. The packet is then tunneled to its final destination. Tunnel-based layer 3 virtualization is a popular virtualization strategy that has been deployed in many software virtualization systems such as VINI [2]. In the following sections, we describe the implementation details of each component in our system and finally their integration.

3.2 Software Virtual Router Implementation

The virtual routers hosted in the PC are implemented by running Click inside OpenVZ containers. Each OpenVZ container has a set of virtual Ethernet interfaces. A software bridge on the PC performs the mapping between the virtual Ethernet interfaces and the physical Ethernet interfaces located in the PC. To enhance flexibility, Click is run as a user mode program inside the OpenVZ container so that the functions of the software virtual router can be fully customized.

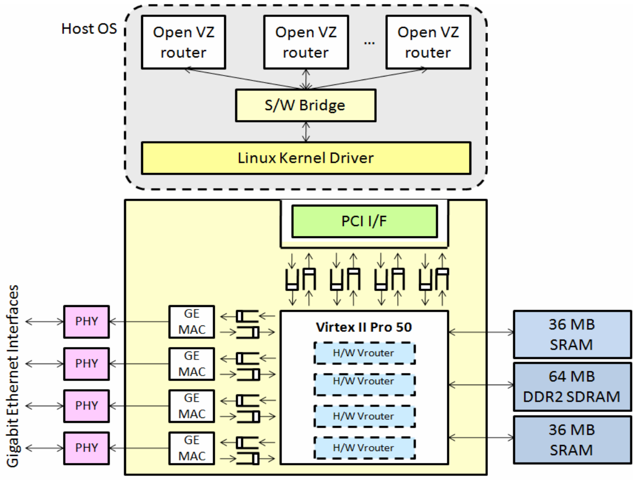

Figure 5. High-level overview of the scalable system

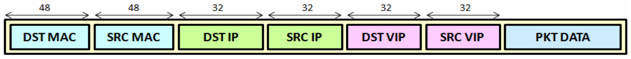

Figure 6 – Packet format for Layer 3 virtualization

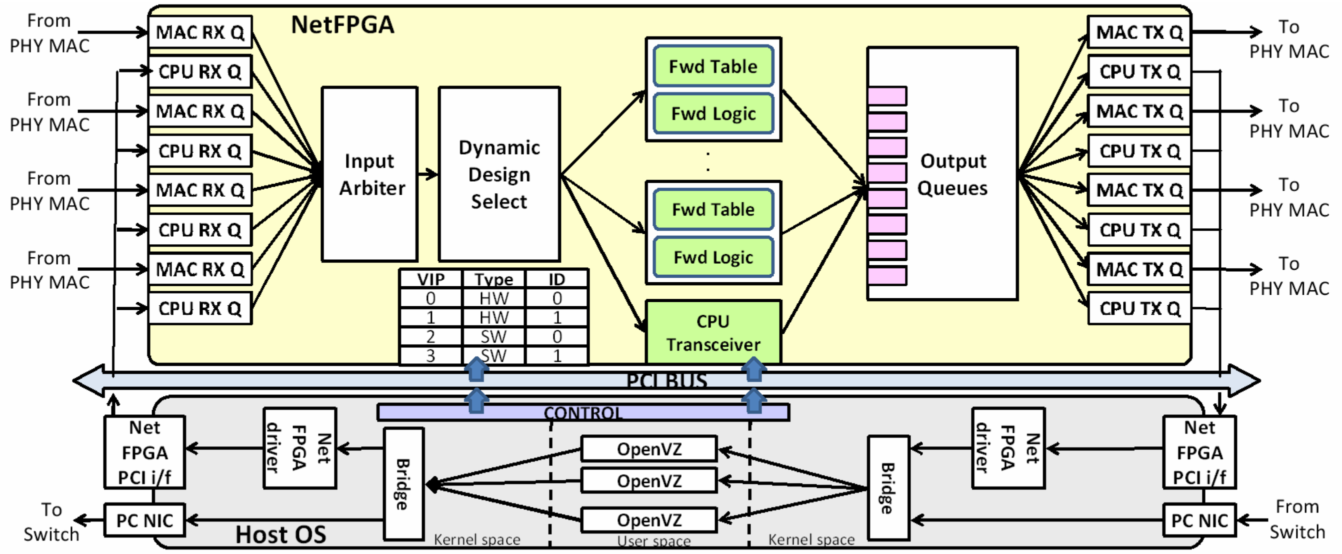

Figure 7 – Detailed system architecture

The penalty of running user mode Click inside the OpenVZ container is that the forwarding speed is much slower. Hence, the software virtual routers should be used in those virtual networks that do not carry too much traffic volume. The software virtual routers are also used to temporarily forward those packets usually forwarded by the hardware virtual routers when the FPGA hardware is being reprogrammed and cannot handle the packets.

3.3 NetFPGA Virtual Router Implementation

The base router [12] provided with the standard NetFPGA distribution has been extended to support multiple, customizable, virtual data planes. The following base router features have been retained for use with our extended router. The hardware data path of the base router consists of input queues, an input arbiter, an output port lookup module, and output queues. Incoming packets from PHY Ethernet interfaces are placed into input queues. The input arbiter module services each queue in a round robin fashion. The output port lookup module consists of SRAM-based forwarding tables that support IP lookup and ARP lookup mechanisms. Processed packets are sent to the output queues from where they are forwarded to the physical interface. The control plane for the base router is implemented in host software running the Linux operating system. The control plane currently supports a modified OSPF (PW-OSPF) routing protocol. Figure 7 shows our extended NetFPGA router architecture which supports four hardware virtual routers and an interface to additional software virtual routers. In this implementation, high speed data planes are constructed in hardware by replicating the forwarding tables and the forwarding logic (output port lookup module) of the base router. Each virtual router has its own unique set of forwarding table control registers. This architecture offers two advantages. First, it ensures close to line rate data plane throughput for each virtual data plane. Second, independent hardware resources facilitate strong isolation between the networks. By providing unique forwarding engines to each virtual router, the system allows network users to customize their data planes independently. Three different forwarding policies are supported: source-based, destination-based and source-and-destination-based routing approaches.

A packet arriving at a PHY input queue (MAC RX Q) undergoes a series of forwarding steps. After selection by the input arbiter in a round robin fashion, a determination is made as to whether the appropriate virtual router is located on the NetFPGA board or in software. The *dynamic design select module* associates each incoming packet with a hardware or software virtual router. The dynamic design select module uses the destination virtual IP address (DST VIP in Figure 6) as an index into a lookup table to determine the associated virtual router. The table is realized using CAM memories in the FPGA. If a match to a hardware virtual router is found, the packet is sent to the virtual hardware forwarding engine which includes a forwarding table and forwarding logic. The forwarding table maps the virtual destination IP address to the next hop virtual destination IP address. The forwarding logic rewrites the source and destination IP addresses (SRC IP and DST IP in Figure 6) of the packet before placing the packet in one of the available output queues. The packet is dispatched to the physical interface via a transmit queue (MAC TX Q). The network administrator can use a programmable register in the NetFPGA card to write entries into the forwarding tables via the PCI bus.

3.4 System Scalability Using Software

The NetFPGA serves as a suitable platform to implement high performance hardware virtual routers. Nevertheless, the choice of a hardware platform for virtualization does create scalability issues. For example, the limited amount of logic available on the Virtex II Pro limits the maximum number of virtual hardware data planes to 5, a sharp reduction from the number of data planes previously implemented using software-based network virtualization. As a result, our system allows for the implementation of virtual routers in both hardware and software.

Two separate, independent approaches are considered to support the implementation of a scalable system. In the first approach, all packets initially enter the NetFPGA card. Packets targeted for virtual networks implemented in software are then forwarded to the PC via the PCI bus. Following packet processing using Click, the packets were returned to the NetFPGA card for retransmission. This approach is subsequently referred to as the **single receiver** approach. In the second **multi-receiver** approach, only packets targeted for NetFPGA routing are received by the NetFPGA card. Packets destined for software virtual routing are received and subsequently retransmitted by a separate PC network interface card (NIC) (Figure 7). Details of each approach are now described.

Single-receiver approach: If an incoming packet does not have a match for a hardware virtual router in the *dynamic design select* table on the FPGA, the packet is sent to the CPU transceiver module shown in Figure 7. The CPU transceiver examines the source of the packet and places the packet in one of the CPU DMA queues (CPU TX Q) interfaced to the host system through the PCI interface. The CPU DMA queues are exposed to the host OS as virtual Ethernet interfaces. The kernel software bridge forwards the Ethernet packet to its respective OpenVZ container based on its destination layer 2 address (DST MAC in Figure 6). The Click modular router within the OpenVZ container processes the packet by modifying the same three packet fields as the hardware router (DST VIP, SRC IP, and DST IP). The software bridge then sends the packet to a CPU RX Q on the NetFPGA board via the PCI bus. After input arbitration, the processed packet is sent back to the CPU transceiver after a *dynamic design select table* lookup. The CPU transceiver module extracts the source and exit MAC queue information from the processed packet. Finally, the packet is placed in the output MAC queue interface (MAC TX Q) for transmission.

The software interface enables migration of virtual networks from software to hardware and vice versa on the fly. A network can be dynamically migrated from hardware to software in three steps. In the first step, an OpenVZ virtual environment that runs the Click router is initiated in the host operating system. Next, all the hardware forwarding table entries are copied to the forwarding table of the host virtual environment. The final step writes an entry into the *dynamic design select table* indicating the association of the virtual IP with software. Our current implementation imposes certain restrictions on virtual network migration from software to hardware. If the software virtual router has a forwarding mechanism that is unavailable in any of the hardware virtual routers, network migration to hardware requires reconfiguration of the FPGA as described in Section 3.5.

Multi-receiver approach: Since in this approach packets can arrive at two different destinations based on their virtual network, network switches are required, as shown in Figure 8. Layer 2 addressing is used to direct each packet to the appropriate destination (NetFPGA card or PC NIC). When deployed in the Internet, we assume that that the sender is capable of classifying each packet as targeted to either the NetFPGA card or PC NIC based on the virtual layer 3 address. This approach requires the use of external hardware (the switches) but simplifies the NetFPGA FPGA hardware design since all packets arriving at the NetFPGA card are processed locally on the card and CPU RX Q and CPU TX Q ports are unused. Packets targeted for software arrive at the PC NIC.

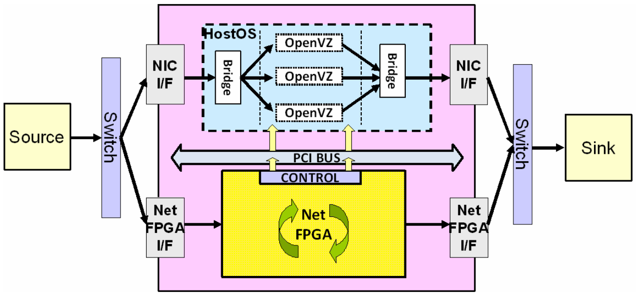

Figure 8. Multi-receiver setup for scalable virtual networking including dynamic FPGA reconfiguration

3.5 Dynamic Reconfiguration

Although forwarding table updates provide some flexibility for virtual routers, it may not be sufficient for significant protocol changes related to the forwarding logic. For example, virtual networks may require a change between source-based, destination-based and source-and-destination based routing. FPGAs provide a unique opportunity for real-time customization of both the forwarding tables *and* the underlying forwarding logic. To illustrate this concept, experiments involving dynamic NetFPGA reconfiguration were performed. All virtual networks remained fully active in software during this reconfiguration by implementing all virtual networks in OpenVZ containers and using the PC NIC (Figure 8) for the actual packet transmission.

Significant changes to the hardware virtual networks can be accommodated by FPGA reconfiguration via the following steps.

1. Software Click instances of all active hardware virtual routers are created in the OpenVZ virtual environment.
2. The Linux kernel sends messages to all nodes attached to the network interface requesting a remap of layer-3 addresses targeted at the NetFPGA board to layer-2 addresses of the PC NIC. Each virtual network includes a mechanism to map between layer-2 and layer-3 addresses. When a virtual network uses IP, the Address Resolution Protocol (ARP) is used to do the mapping between layer-2 and layer-3 addresses. In our prototype, where IP is used in the data plane, the ARPFaker element [29] implemented in Click is used to generate ARP reply messages to change the mapping between layer-2 and layer-3 addresses.
3. Once addresses are remapped, all network traffic is redirected to the PC for forwarding with software virtual routers.
4. The FPGA is completely reprogrammed with a new bitstream that incorporates changes in network characteristics. In our implementation, a collection of

previously-compiled FPGA bitstreams is used. Partial reconfiguration of selected network components in the FPGA will be addressed in future work.

5. The routing tables are written back to the hardware following reconfiguration.
6. In a final step, the Linux kernel sends messages to all nodes attached to the network interface requesting a remap of layer-3 addresses back to the NetFPGA interface. The virtual network then resumes operation in the hardware data plane for the instantiated hardware routers.

In Section 5, the overheads of this dynamic reconfiguration approach are quantified.

3.6 Power Optimization

The use of programmable hardware offers the opportunity to disable the power consumption of individual hardware virtual routers when they are not needed. To allow for additional flexibility, the forwarding table and forwarding logic of individual hardware virtual routers were instrumented with clock gating circuitry. A programmable register for each router can be set to disable the clock for all logic registers and BRAM blocks in the selected forwarding table and logic.

4. EXPERIMENTAL APPROACH

The following sections describe some of the techniques used to obtain experimental results.

4.1 System Parameters

To measure performance of hardware virtual routers, we use the network configuration shown in Figure 9. Initial experiments explored the performance of the hardware virtual routers. For these experiments, the NetFPGA hardware packet generator and packet capture tool [6] were used to generate packets at line rate. For all additional experiments which included the software virtual routers, iPerf [31] was used to generate packets. In general, the software routers cannot handle line rate traffic. The hardware virtual router implementations were compared against the Click modular router running on a Linux box which includes a 3 GHz AMD X2 6000+ processor, 2 GB of RAM, and a dual-port Intel E1000 Gbit Ethernet NIC in the PCIe slot. Performance was determined in terms of network latency and observed throughput. The network latency was measured with the *ping* utility.

To measure the scalability of our system, systems of between 1 and 15 virtual routers were implemented. For these systems, between one and four virtual routers were implemented in hardware while the rest were implemented in host software. The Synopsys VCS simulator was used for the functional verification of the hardware designs. The Xilinx XPower (XPE) power estimator was used to determine power estimates. All hardware designs were successfully deployed on a NetFPGA cube and executed on the Virtex II FPGA.

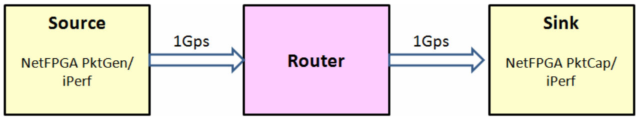

Figure 9 – Test topology

5. EXPERIMENTAL RESULTS

5.1 Performance

In an initial experiment, the baseline performance of a single hardware virtual router running in the NetFPGA hardware and a Click software virtual router running in the OpenVZ container are compared. Figure 9 shows the testbed network used in our experiments. The NetFPGA packet generator/capture tools were used to generate traffic of different packet sizes and rates. The packet generator was loaded with PCAP files [6] with packet sizes of 64 to 1024 bytes. Packets were transmitted to our system at the line rate of 1Gbps.

The throughput of three specific system configurations was considered:

1. *Hardware data plane* – network traffic is received and transmitted by the NetFPGA board and forwarding is performed using a hardware virtual router on the NetFPGA board.
2. *Click/OpenVZ from NIC* – network traffic is received and transmitted by the PC NIC (Figure 7) network interfaces and forwarding is performed using a Click router in OpenVZ.
3. *Click/OpenVZ from NetFPGA* – network traffic is received and transmitted by the physical NetFPGA network interfaces but the forwarding operations are performed using a Click router in OpenVZ. Packets are transferred between the NetFPGA hardware and OpenVZ over the PCI bus.

The throughput of the three approaches for differing packet sizes is shown in Figure 10. These values show the maximum achievable throughput by each implementation for a packet drop rate of no more than 0.1% of transmitted packets. The receiver throughputs were measured by hardware counters in the NetFPGA PktCap capture tool (Figure 9).

The throughput of shorter packets drops considerably in the software-based implementations. In contrast, the single hardware virtual router consistently sustains throughputs close to line rates for all packet sizes. The hardware provides one to two orders of magnitude better throughput than the OpenVZ Click router implementions due to inherent inefficiencies in the software implementation. The OpenVZ running in user space trades off throughput for flexibility and isolation. The performance degradation in software implementations results from frequent operating system interrupts and system calls during packet transfers between user space and kernel space. The hardware virtual routers takes advantage of the parallelism and specialization of the FPGA to overcome these issues.

5.2 Network Scalability

Network scalability can be measured in terms of both throughput and latency. For these experiments, the test topology was configured as shown in Figure 9. Four specific system configurations were considered for systems that required between 1 and 15 virtual networks. The software-only *Click/OpenVZ from NIC* and *Click/OpenVZ from NetFPGA* cases are the same as defined in Section 5.1.

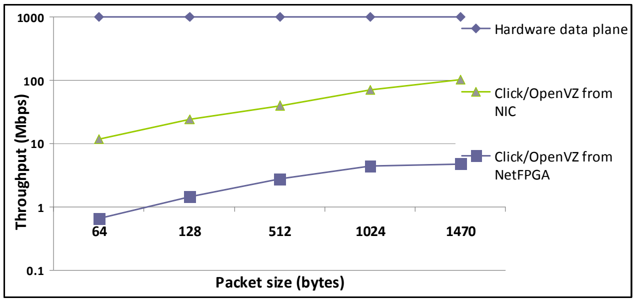

Figure 10. Receiver throughput versus packet size for a single virtual router

Additional cases which combine NetFPGA and software forwarding include:

- *Hardware+Click/OpenVZ from NIC* - network traffic targeted to virtual networks implemented in software is received and transmitted by the PC NIC network interfaces and forwarding is performed using a Click router in OpenVZ. Network traffic targeted to virtual networks implemented by the NetFPGA is received and transmitted by the NetFPGA. This case represents the **multiple receiver** approach described in Section 3.4.
- *Hardware+Click/OpenVZ from NetFPGA* - all network traffic is received and transmitted by the physical NetFPGA network interfaces. Some forwarding operations are performed by hardware virtual routers while others are performed using Click routers in OpenVZ. For the latter cases, packets are transferred between the NetFPGA hardware and OpenVZ over the PCI bus. This case represents the **single receiver** approach described in Section 3.4.

In the latter two cases, up to four virtual routers were implemented in hardware and remaining networks (up to 11) were implemented in software (Click/OpenVZ).

Transmission latency for the four cases were measured in the presence of background traffic. The packets of the background traffic were transmitted at the maximum capacity available to each router. The packet generation rate was fixed such that no more than 0.1% of the traffic was dropped. As shown in Figure 11, the average network latency of the Click OpenVZ virtual router is approximately an order of magnitude greater than that of the hardware implementation. The latency of OpenVZ increases by approximately 66% from one to fifteen virtual routers. This effect is due to context switching overhead and resource contention in the operating system. Packets routed through OpenVZ via the NetFPGA/PCI interface incur 50% additional latency overhead than when they are routed through the NIC interfaces. The average latency of hardware routers remains constant for up to four data planes. After this, every additional software router increases the average latency by 25%.

To measure aggregate throughput when different numbers of virtual routers are hosted in our system, 64 byte packets were initially transmitted with an equal bandwidth allocated for all networks. Next, the bandwidth share of each virtual network was incrementally increased until the networks began to drop more than 0.1% of the assigned traffic.

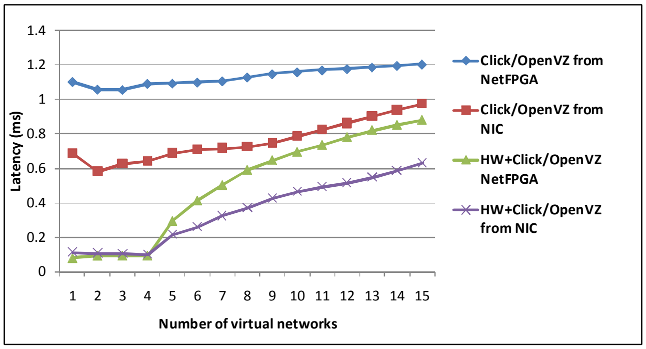

Figure 11 – Average latency for an increasing number of virtual routers

A single OpenVZ software virtual router can route packets through the PC NIC interface at a bandwidth up to 11 Mbps. The throughput drops by 27% when fourteen additional software routers are added. The software virtual router implementation which routes packets from the NetFPGA card to the OpenVZ containers can sustain only low throughput (approximately 800 Kbps) with 64 byte packets and 5 Mbps with 1500 byte packets due to inefficiencies in the NetFPGA PCI interface and driver. The hardware virtual router sustains close to line rate aggregate bandwidths for up to four data planes. The average aggregate bandwidth drops when software virtual routers are used in addition to hardware routers. Note that the use of a log scale causes the top two plots overlap.

The top two plots (*HW+Click/OpenVZ from NIC* and *HW+ Click/OpenVZ from NetFPGA*), which overlap in Figure 12, show the average aggregate throughputs when software data planes are used in conjunction with hardware data planes. Since the hardware throughput dominates the average throughput for these two software data plane implementations, minor differences in bandwidth are hidden. Further, the use of a log scale hides minor differences in throughput between the two software implementations.

Our virtual networking systems which contain more than the four virtual routers implemented in hardware exhibit an average throughput reduction and latency increase as software virtual routers are added. For systems that implement a range of virtual networks with varying latency and throughput requirements, the highest performance networks could be allocated to hardware while lower performing networks are implemented in software.

5.3 Resource usage

The Virtex II Pro FPGA can accommodate a maximum of five virtual routers, each with a 32-entry forwarding table. When the CPU transceiver module is included, the FPGA can accommodate a maximum of four virtual routers. Each virtual data plane occupies approximately 2000 slice registers and 3000 slice LUTs. A fully populated design uses approximately 90% of the slices and 40% of the BRAM. Table 1 shows the resource utilization of up to five virtual routers. All designs are operated at 62.5 MHz. Synthesis results for the virtual router design implemented on the largest Virtex 5 (5vlx330tff1738) shows that a much larger FPGA could support up to 32 virtual routers.

Power consumption results show that a two virtual router system consumes approximately 158.8 mW of static power and 766.2 mW of dynamic power. It is possible to save about 10% of the total dynamic power by clock gating an individual virtual router. Additional power results are summarized in Table 2.

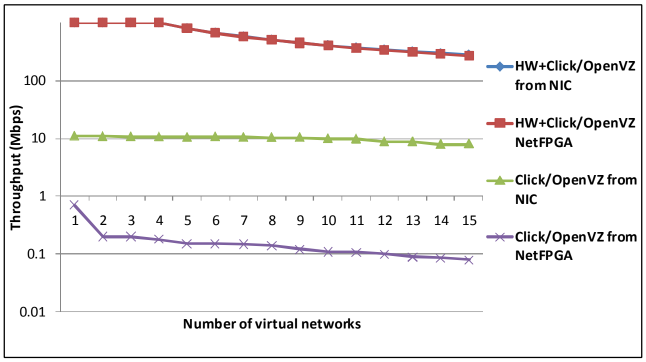

Figure 12 – Average throughput for an increasing number of virtual routers

	Hardware data planes				
	1	**2**	**3**	**4**	**5**
Slices	10068	12882	15696	18509	21322
Slice FF	8964	11269	13574	15879	18184
LUTs	15272	19744	24216	28689	33161
IO	437	437	437	437	437
BRAM	25	40	55	70	85

Table 1 – Resource utilization

#Virtual routers	Static (mW)	Dynamic (mW)	Total (mW)
1	158.8	687.5	846.3
2	158.8	766.3	925.1
3	158.8	853.2	1012.3
4	158.8	936.2	1095.0

Table 2 – Power consumption of virtual routers

5.4 Overhead of Dynamic Reconfiguration

To evaluate the cost and overhead of dynamic reconfiguration, we initially programmed the target FPGA with a bitstream that consisted of a single virtual router. An experiment was then performed in which a source host sent ping packets to the system at various rates which were forwarded using the NetFPGA hardware virtual router. Periodically, the hardware virtual router was migrated to a software-based router running in the OpenVZ container of the PC using the procedure described in Section 3.5. After FPGA reconfiguration, the virtual router was migrated back to the NetFPGA card.

It was determined that approximately 12 seconds are required to migrate a hardware virtual router to a Click router implemented in OpenVZ. The FPGA reconfiguration, including bitstream transfer over the PCI bus, required about 5 seconds. Transferring the virtual router from software back to hardware took only around 3 seconds. The relatively high hardware-to-software migration latency was caused by the initialization of the virtual environment and the address remapping via ARP messages. The software to hardware transfer only requires writes to forwarding table entries over the PCI interface. Our experiments show that if a source generates packets at 10,000 or fewer packets per second, our system can gracefully migrate the virtual router between hardware and software without any packet loss.

6. CONCLUSIONS AND FUTURE WORK

A scalable network virtualization environment that implements fast data planes in hardware and slower data planes in software is presented. The system exploits the parallelism, specialization and adaptability in FPGAs to realize a flexible network virtualization substrate that can be used for realistic network experimentation. In the future, we plan to support partial reconfiguration and automatic scheduling of dynamic reconfiguration in the system. The evaluation of the clock gating scheme as a power efficient fault injection mechanism for network experimentation is a possibility. We also plan to evaluate various algorithms that can efficiently allocate virtual networks with diverse bandwidth requirements on the heterogeneous virtualization substrate. Finally, the efficiency of the system can be improved by integrating efficient kernel-based virtual router implementations in software with fast hardware data planes in much larger FPGAs.

7. ACKNOWLEDGMENTS

The work was funded in part by National Science Foundation grant CNS-0831940. The FPGA compilation tools were generously donated by Xilinx Corporation.

8. REFERENCES

[1] M. B. Anwer and N. Feamster. Building a fast, virtualized data plane with programmable hardware. ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures, pp. 1-8, 2009.

[2] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. SIGCOMM Comput. Commun. Rev., 36(4), pp. 3-14, 2006.

[3] N. Beheshti, J. Naous, Y. Ganjali, and N. McKeown. Experimenting with buffer sizes in routers. In Proceedings: ACM/IEEE Symposium on Architecture for Networking and Communications Systems, pp. 41-42, 2007.

[4] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Hosting virtual networks on commodity hardware. Georgia Tech Computer Science Technical Report GT-CS-07-10, 2008.

[5] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J.

Rexford. “Trellis: A platform for building flexible, fast virtual networks on commodity hardware”. Proc. Workshop on Real Overlays and Distributed Systems, 2008

[6] G. A. Covington, G. Gibb, J. W. Lockwood, and N. McKeown, A packet generator on the NetFPGA platform. Proceedings of the 17th IEEE Symposium on Field-Programmable Custom Computing Machines, pp.235-238, 2009.

[7] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley. Evaluating xen for router virtualization. Proceedings of 16th International Conference on Computer Communications and Networks, pp. 1256-1261, 2007.

[8] N. Feamster, L. Gao, and J. Rexford. How to lease the Internet in your spare time. ACM SIGCOMM Computer Communication Review, 37(1), pp. 61-64, 2007.

[9] R. Franklin, D. Carver, and B. Hutchings. Assisting network intrusion detection with reconfigurable hardware. In Proceedings: IEEE International Symposium on Field Programmable Custom Computing Machines, pp. 111-120, 2002.

[10] F. Kuhns, J. DeHart, A. Kantawala, R. Keller, J. Lockwood, P. Pappu, D. Richards, D. Taylor, J. Parwatikar, E. Spitznagel, J. Turner, and K. Wong. Design of a high performance dynamically extensible router. In Proceedings: DARPA Active Networks Conference and Exhibition, pp. 42-64, 2002.

[11] A. Menon, A. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In Proceedings: USENIX Annual Technical Conference, pp. 15-28, 2006.

[12] Stanford University. NetFPGA User’s Guide, 2008. http://yuba.stanford.edu/NetFPGA/static/guide.html.

[13] J. Turner. A proposed architecture for the GENI backbone platform. WUCSE2006-14 Washington University School of Applied Science and Engineering Technical Report, 2006.

[14] G. Watson, N. McKeown, and M. Casado. NetFPGA: a tool for network research and education. In Proceedings: Workshop on Architectural Research Using FPGA Platforms, pp. 160-161, 2006.

[15] N. Weaver, V. Paxson, and J. Gonzalez. The Shunt: An FPGA-based accelerator for network intrusion protection. In Proceedings: ACM/SIGDA International Sympoosium on Field Programmable Gate Arrays, pp. 199-206, 2007.

[16] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, “ROFL: routing on flat labels,” SIGCOMM Comput. Commun. Rev., 36(4), pp. 363–374, 2006.

[17] J. Turner et al, “Supercharging PlanetLab: a high performance, multi-application, overlay network platform,” in Proceedings of SIGCOMM, pp. 85-96, 2007.

[18] L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization", Proceedings of the Third Workshop on Hot Topics in Networking, pp. 34-41, 2004.

[19] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The Second-Generation Onion Router", Proceedings of the 13th USENIX Security Symposium", pp. 303-320, 2004.

[20] L. Zhuang, F. Zhou, B. Zhao and A. Rowstron, "Cashmere: Resilient Anonymous Routing", Proceedings of NSDI, pp. 301-314, 2005.

[21] E. Keller and E. Green, "Virtualizing the data plane through source code merging.” in Proc. ACM SIGCOMM Workshop on Programmable Routers for the Extensible Services of Tomorrow, pp. 9-14, 2008.

[22] G. Lu, Y. Shi, C. Guo, and Y. Zhang, " CAFE: A Configurable pAcket Forwarding Engine for Data Center Networks” in Proc. ACM SIGCOMM Workshop on Programmable Routers for the Extensible Services of Tomorrow, pp. 25-30, 2009.

[23] OpenVZ project page, http://www.openvz.org, 2008

[24] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive", SIGCOMM Comput. Commun. Rev., vol. 38, no. 4, pp. 231-242, 2008.

[25] Y. Liao, D. Yin and L. Gao, “PdP: Parallelizing Data Plane in Virtual Network Substrate", Proceedings of the First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, pp. 9-18, 2009

[26] VMWare, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist", 2007, http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf"

[27] A. Whitaker, M. Shaw, and S. Gribble, "Denali: Lightweight Virtual Machines for Distributed and Networked Applications, Proceedings of the USENIX Annual Technical Conference", 2002.

[28] S. Soltesz, H. Potzl, M. Fiuczynski, A. Bavier and Larry Peterson, “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors”, Proceedings of the ACM SIGOPS European Conference on Computer Systems, pp. 275-287, 2007.

[29] “The Click Modular Router”, http://read.cs.ucla.edu/click/

[30] “Tunneling”, www.linuxfoundation.org/en/Net:Tunneling

[31] “Iperf 1.7.0: The TCP/UDP bandwidth measurement tool.” http://dast.nlanr.net/Projects/Iperf/