Reconfigurable Data Planes for Scalable Network Virtualization

Deepak Unnikrishnan, Ramakrishna Vadlamani, Yong Liao, Jérémie Crenne, Lixin Gao, *Fellow, IEEE,* and Russell Tessier, *Senior Member, IEEE*

Abstract—Network virtualization presents a powerful approach to share physical network infrastructure among multiple virtual networks. Recent advances in network virtualization advocate the use of field-programmable gate arrays (FPGAs) as flexible high performance alternatives to conventional host virtualization techniques. However, the limited on-chip logic and memory resources in FPGAs severely restrict the scalability of the virtualization platform and necessitate the implementation of efficient forwarding structures in hardware. The research described in this manuscript explores the implementation of a scalable heterogeneous network virtualization platform which integrates virtual data planes implemented in FPGAs with software data planes created using host virtualization techniques. The system exploits data plane heterogeneity to cater to the dynamic service requirements of virtual networks by migrating networks between software and hardware data planes. We demonstrate data plane migration as an effective technique to limit the impact of traffic on unmodified data planes during FPGA reconfiguration. Our system implements forwarding tables in a shared fashion using inexpensive off-chip memories and supports both Internet Protocol (IP) and non-IP based data planes. Experimental results show that FPGA-based data planes can offer two orders of magnitude better throughput than their software counterparts and FPGA reconfiguration can facilitate data plane customization within 12 seconds. An integrated system that supports up to 15 virtual networks has been validated on the NetFPGA platform.

Index Terms—Internetworking routers, Design, Virtual networks, FPGA

1 Introduction

The Internet provides a critical infrastructure for a wide variety of end user services. These services, which range from gaming and social networking to high-end business applications, require widely varying quality-of-service, robustness, and performance parameters [1]. It is infeasible to support distinct networking resources for each service due to practical and economic constraints. Instead, it is desirable to share the physical network infrastructure and allocate resources according to individual service requirements. As the demand for new services and higher performance grows, the network core must fundamentally adapt to support emerging protocols and architectures. Unfortunately, the architecture and scale of the modern Internet continues to threaten its evolution.

Network virtualization addresses the adaptability issue by allowing multiple virtual networks to operate over a shared physical infrastructure. Each virtual network can run custom routing protocols and support diverse data plane architectures, facilitating incremental deployment of novel networking techniques on legacy architectures. In addition, infrastructure providers can take advantage of the shared approach to improve network utilization and boost revenues.

Flexibility, high performance and ease-of-use are critical to the wide-scale deployment of network virtualization systems. To date, most network virtualization research has focused on optimized software-only approaches. Software techniques use host or container virtualization to create virtual routing instances [2] [3]. Despite being flexible and easy to deploy, such approaches only offer limited network throughput and suffer from increased packet latencies due to the sequential nature and limited memory bandwidth of off-the-shelf microprocessors. This constraint naturally motivates the need for a hardware-based solution.

Recent FPGA-based implementations of virtual network data planes [4] [5] show improved performance. However, the limited logic and memory resources of FPGAs severely constrain the number of data planes supported by the system. The implementation of forwarding structures, such as forwarding tables using on-chip memory, in these approaches create scalability issues. Once programmed, FPGA-based data plane behavior is largely fixed, even though the SRAM-based FPGA could be reconfigured to support changing virtual network needs. FPGA reconfiguration, however, can affect traffic in virtual data planes which do not require updates, as all or portions of the FPGA device may be unavailable for a period of time while reconfiguration is performed. In many cases, existing host virtualization techniques running in software can be sufficient to implement affected virtual networks while reconfiguration occurs.

- *D. Unnikrishnan, L. Gao and R. Tessier are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, 01003 USA e-mail: (tessier@ecs.umass.edu).*
- *R. Vadlamani is with Qualcomm Inc., Boxborough, MA, 01719 USA*
- *Y. Liao is with Microsoft Corporation, Seattle, WA, 98052 USA*
- *J. Crenne is with Université de Bretagne Sud, Lorient 56100 France*

The limitations of FPGA and software-only netwok virtualization approaches motivate us to consider a *heterogeneous* and *adaptive* approach to assigning virtual networks to hardware and software resources based on service requirements.

A major research contribution of this work is a new network virtualization platform that addresses previous scalability and performance issues. The system integrates multiple virtual data planes implemented in FPGA hardware with additional software data planes hosted by a PC kernel. Virtual networks with the most stringent performance requirements are assigned to data planes implemented in FPGA hardware while lower performance networks are assigned to virtual data planes in host software. The hardware data planes use an optimized hardware architecture that stores forwarding tables from multiple virtual data planes in a shared fashion using off-chip SRAM memories. When virtual network performance requirements change, the system adapts itself by dynamically swapping virtual networks between software and hardware virtual data planes. We use FPGA reconfiguration to aid virtual data plane migration. During dynamic FPGA reconfiguration, virtual data planes implemented in hardware can continue to transmit packets via virtual data planes in software. The same networking protocols and data plane isolation approaches can be applied to both platforms in a fashion which is hidden from the system user.

We evaluate the throughput and latency of the network virtualization platform using a NetFPGA card [6]. Our approach has been validated using IP-based (IPv4 router) and non-IP based data planes (Routing on Flat Labels (ROFL) [7] router). The hardware-based IPv4 data plane uses longest prefix matching tables implemented in external SRAM memories to forward packets. These forwarding tables can support up to 512K entries by sharing 4.5 MBytes (MB) of off-FPGA SRAM memory. We implement the hardware data planes in the Virtex II FPGA located on the NetFPGA card and evaluate them at line rates. Software data planes are integrated into the OpenVZ virtualization environment [8] running on an off-the-shelf personal computer (PC). The system supports virtual network migration between hardware and software data planes within 12 seconds. The Virtex II FPGA supports a total of five concurrent virtual data planes. Each of these data planes support line rate throughput (1 Gbps).

2 Background

Figure 1 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 table stores the routing entries and performs packet forwarding. The virtual routers are independent of each other and can run

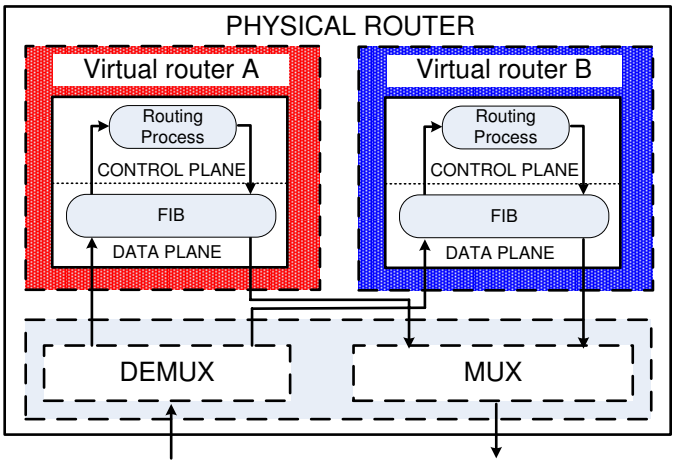

Fig. 1. Virtual router architecture. Each data plane contains a forwarding information base (FIB).

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 a color (e.g. red or blue) and data packets belonging to each virtual network 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, they are colored with the router's color at the MUX before they enter 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 (e.g. ROFL [7] [9]) requires customization of nearly all aspects of the network core, such as the routing protocol and the address lookup algorithm. Other examples include quality of service (QoS) schemes that require certain queuing and scheduling approaches and security mechanisms such as network anonymity and onion routing [10] [11]. To attract existing applications whose performance requirements may scale over time, superior data plane performance is of utmost importance. In addition, the virtualization platform must also scale to support hundreds of simultaneous virtual networks while catering to their dynamic performance requirements.

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 em-

ploy application-specific integrated circuits (ASICs) [12]. Although ASICs typically achieve high performance, they do not have the necessary flexibility needed for service customization. For example, the Supercharging PlanetLab platform [13] only provides a customizable forwarding table interface, which makes it hard to support innovations such as network anonymity, onion routing and QoS schemes. Additionally, ASICs incur long design cycles and mask costs, making them prohibitively expensive to prototype new architectures.

Several network virtualization systems that use host virtualization techniques have been proposed [3] [14] [15] [16] [17]. A comprehensive survey of virtual network implementations using software techniques can be found in [18]. In general, software-based data plane implementations demonstrate a higher degree of flexibility than ASICs. However, they suffer from lower operating throughput and higher statistical variations in observed network parameters due to jitter and resource contention [2].

The evaluation of network virtualization using FPGAs is much more limited than previous software efforts. Anwer et al. [4] [19] demonstrate the implementation of up to 8 virtual data planes in a single Virtex II Pro on a NetFPGA board. Although this setup has been shown to provide twice as much throughput as a software kernel router, a number of limitations exist. For example, the logic resources of the FPGA impose a hard cap on the number of supported virtual networks. The hardware data planes further use FPGA on-chip memories that can only store a limited number of forwarding table entries, limiting the scalability of the overall system. In addition, the dynamic reconfiguration abilities of the FPGA could be used to customize data planes according to changing virtual network needs.

CAFE [5] implements a similar platform that supports distinct virtual data planes on the NetFPGA. A salient feature of the CAFE architecture is the presence of user configuration registers that allow real-time updates to virtual routing table protocols. However, like previous approaches, CAFE presents scalability issues and offers limited ways to customize the properties of the virtual data planes.

The architecture of forwarding tables is another important design consideration for FPGA-based virtual data planes. Typical forwarding tables need to store hundreds of thousands of entries and consume significant memory resources within the FPGA. Although the design of efficient forwarding tables for general-purpose (e.g. non-virtualized) routers has been well researched in the past [20] [21], recent advances in network virtualization have inspired researchers to revisit this problem in the context of network virtualization. Specifically, the virtualization platform must efficiently store forwarding entries from multiple virtual routers in a shared fashion using inexpensive off-chip memories. Fu et al. [22] and Song et al. [23] describe approaches to compact virtual forwarding tables into a single data structure based on binary *tries*. Although trie-based approaches are attractive for software-based forwarding table implementations, practical hardware designs require heavily pipelining to achieve high packet forwarding throughput. Additionally, the hardware cost of trie-based techniques exponentially grows with longer prefix lengths. In contrast, we adopt an approach that enables hardware data planes to store forwarding table entries in inexpensive off-chip SRAMs without the need for heavy pipelining in hardware.

3 System Design

Our system makes three specific contributions to existing network virtualization platforms:

1) We present a *heterogeneous* virtualization platform that combines fast hardware data planes implemented in FPGAs with slower software data planes implemented using host virtualization techniques. The heterogeneity in virtualization resources is used to scale the number of data planes beyond the logic capacity of pure FPGA-based virtualization platforms. We validate this system using both IP and non-IP based data planes.
2) The system *adapts* to cater to the changing virtual network service requirements by dynamically migrating active virtual networks between hardware and software data planes. FPGA reconfiguration is used to aid data plane migration. During FPGA reconfiguration, unmodified hardware data planes can be temporarily migrated to software so that they can continue to transmit traffic.
3) To promote scalability, the system implements an optimized hardware data plane architecture that stores forwarding tables from multiple virtual routers in a shared fashion using inexpensive off-chip SRAM memories. The architecture obviates the need for heavy pipelining in hardware.

In the following subsections, we present the major design goals, an overview of the architecture, details of the hardware and software data planes and strategies to scale the data planes.

3.1 System Overview

Our design decisions are driven by two design goals. The primary design goal of the system is to improve the scalability of existing homogeneous FPGA-based network virtualization platforms. The scalability limitations in existing FPGA-based platforms originate from two factors. First, the limited logic resources (slices, flip flops etc.) constrain the number of simultaneous data planes that can operate on the device. Simply increasing the FPGA size to scale the number of data planes is cost-inefficient since FPGA cost generally does not scale linearly with device capacity. Second, individual hardware data planes use *separate* on-chip memory resources, such as block RAMs (BRAMs) and TCAMs, to store forwarding tables.

Such an implementation does not scale well with larger forwarding tables or a greater number of data planes. It is therefore important to scale both the number of data planes *and* the size of forwarding tables to build a practical network virtualization platform.

The secondary design goal of the architecture is to improve the design flexibility of hardware data planes through FPGA reconfiguration. Although FPGAs offer high data plane design flexibility by virtue of their reconfiguration properties, customization of individual hardware data planes in the same FPGA through static reconfiguration additionally requires that traffic in active virtual networks, other than the one being customized, be stopped during the reconfiguration procedure. It is therefore necessary for the architecture to support hardware data plane customization with minimal traffic disruption for unmodified hardware data planes.

Our architecture includes hardware and software techniques to address these design goals. Specifically, we implement additional virtual data planes in host software using container virtualization techniques to scale the number of data planes beyond the logic capacity of the FPGA (Section 4). We address the limitations in memory scalability by implementing forwarding tables from multiple hardware data planes in a *shared* fashion using inexpensive external SRAM memories located outside the FPGA (Section 5). The architecture enables customization of hardware data planes using *virtual network migration* between hardware and software when virtual networking requirements change.

The high-level architecture of our system built on the NetFPGA [24] platform is shown in Figure 2. In this system, virtual data planes that require the highest throughput and lowest latency are implemented on a Virtex II-Pro 50 FPGA on the NetFPGA while additional software virtual data planes are implemented in OpenVZ containers running on the PC. The reference NetFPGA platform consists of four 1 Gbps Ethernet interfaces, a 33 MHz PCI interface, 64 MB of DDR2 DRAM and two 18-Mbit SRAMs. The hardware data path, implemented within the FPGA, is organized as a pipeline of fully customizable modules. The forwarding tables of the hardware virtual data planes can either be implemented using BRAM and SRL16E blocks within the FPGA or using the 36-Mbit SRAM located external to the FPGA. In either case, forwarding tables can be updated from software through the PCI interface. This 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 data planes. The PC server is sliced into virtual machines using OpenVZ. The OpenVZ framework is a lightweight virtualization approach used in several network virtualization systems [15] [25] and it is included in major Linux distributions. The OpenVZ kernel allows multiple isolated user-space instances, which are called virtual machines or containers. Data planes can be spawned in host software when an FPGA can no longer accommodate new data planes. Since software virtual data planes must be effectively isolated from each other, they are hosted in isolated OpenVZ containers. The OpenVZ virtual environment guarantees that the each container gets a fair share of CPU cycles and physical memory. Each instance of the OpenVZ container executes a user mode Click modular router [26] to process the packets. The forwarding functions of Click can be customized according to the virtual network creator's preferences.

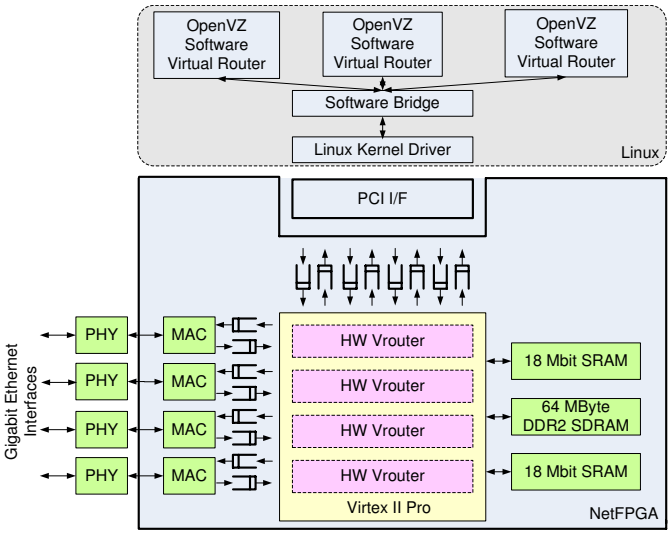

Fig. 2. High level architecture

Packet forwarding operates as follows. When a packet arrives at an Ethernet interface (PHY), the destination address in the packet header is used to determine the location of its data plane. If the packet is associated with a virtual network hosted in the FPGA, it is processed by the corresponding hardware data plane. 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-based data planes. Periodically, the virtual network administrator can reconfigure virtual networks in the FPGA to take changes in bandwidth demands and routing characteristics into account. While the FPGA is being reconfigured, all traffic is routed by the host software.

Next, we describe the detailed architecture of FPGA-based and software-based data planes.

3.2 Hardware Data Planes

The hardware data planes of our virtualization platform are constructed by customizing NetFPGA's modular datapath [6], as shown in Figure 3. We retain the basic components of the datapath including input queues, input arbiter and output queues. Besides these standard components, the system includes two additional hardware modules. The *dynamic design select* module provides the demux interface to virtual data planes in hardware for packets arriving at the physical network interfaces. The *CPU transceiver* module facilitates transmission of packets to virtual data planes in host software.

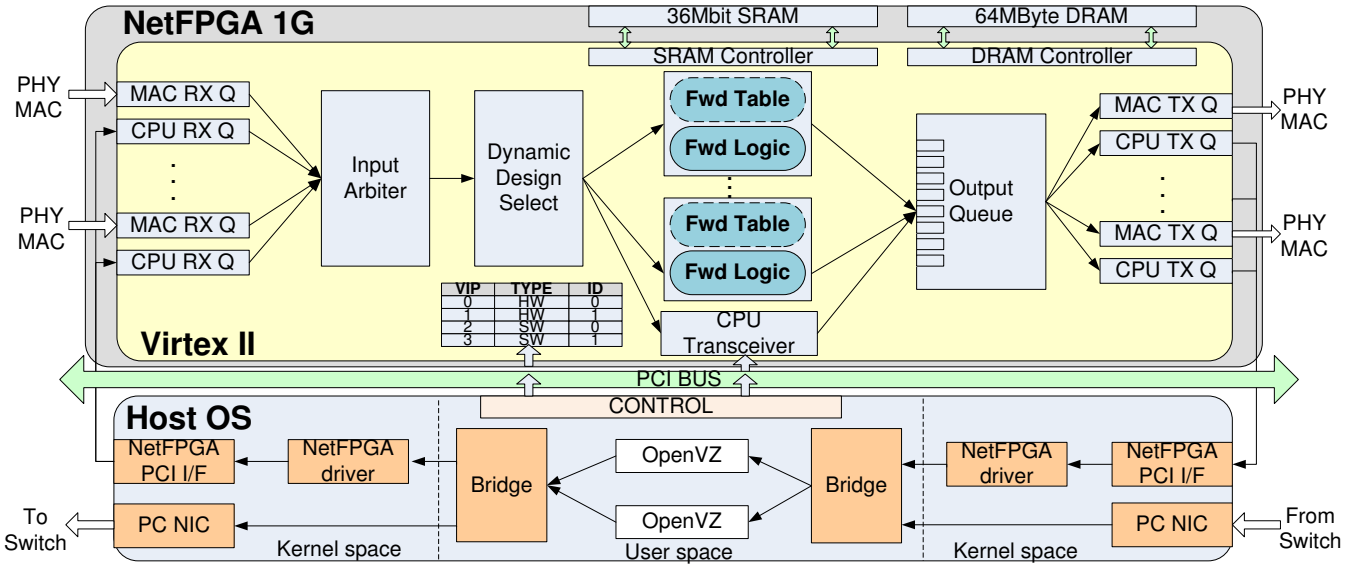

Fig. 3. Detailed system architecture

When a packet enters the system, it is automatically classified by the dynamic design select module based on the virtual destination address in the packet header. Packets belonging to virtual networks can be classified based on virtual IP addresses or virtual MAC addresses in packet headers. The mapping from virtual networks to virtual data planes can be programmed into the *dynamic design select table* using NetFPGA's register interfaces by a person administering virtual networks (hereafter referred to as the operator). The CPU transceiver module provides an interface to transmit and receive packets from virtual data planes in host software. More details regarding the operation of the CPU transceiver module are provided in Section 4.

We implement the forwarding logic of hardware data planes by customizing instances of the output port lookup module [6], which encapsulates the forwarding logic of the NetFPGA reference router. Each virtual data plane 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, by reserving independent hardware resources such as logic slices, registers and memory bits for each virtual router, the architecture eliminates the need for resource sharing, facilitating strong isolation.[1]

Forwarding tables of individual data planes are implemented using TCAM or BRAM resources within the FPGA or using SRAM memories located outside the FPGA. When forwarding tables are implemented using on-chip memory, the forwarding logic integrates TCAMs that support IP and Address Resolution Protocol (ARP) lookup mechanisms. Section 5 describes the implementation of forwarding tables using external SRAMs. When forwarding tables are stored in external SRAMs, input and output queues must be implemented using the DDR2 DRAM memory. We implement the control planes for the virtual networks hosted in the FPGA in host software using the Linux operating system. The control planes currently support a modified OSPF (PW-OSPF) routing protocol.

1. The placement of data plane logic is not spatially constrained within the FPGA. Although routing wires may overlap between the data planes, the data planes do not contend for routing resources.

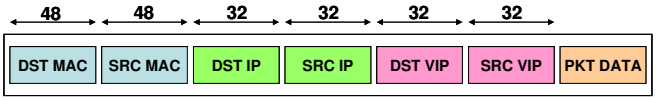

Fig. 4. Packet format for layer 3 virtualization

Figure 3 shows the architecture of a virtualization platform which supports four hardware virtual data planes and an interface to additional software data planes. The hardware data planes in this example support both IP and non-IP based forwarding techniques. The IP-based data planes support source-based, destination-based and source-and-destination-based routing approaches. The non-IP data plane forwards packets based on ROFL [7], a flat label lookup. We present the details of these data planes next.

3.2.1 IPv4

The IPv4 data plane design example uses layer 3 virtualization based on IPIP tunneling. Tunneling transforms data packets into formats that enable them to be transmitted on networks that have incompatible address spaces and protocols. In this tunneling approach, the network operator assigns a virtual IP address from a private address space to each node in a virtual network. To transmit a packet to another virtual node in the private address space, the source node encapsulates the packet data in a virtual IPv4 wrapper and tunnels it through intermediate routers. When the packet reaches

a virtual node, the data plane uses an inner virtual IP address 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].

The dynamic design select module uses the destination virtual IP address (DST VIP in Figure 4) as an index into the design select table to determine the associated data plane. If a match to a virtual network in the FPGA is found, the dynamic design select module sends the packet to the hardware plane. The forwarding engine maps the virtual destination IP address to the next hop virtual destination IP address and rewrites the source and destination IP addresses (SRC IP and DST IP in Figure 4) of the packet before forwarding the processed packet through output queues.

3.2.2 ROFL

ROFL [7] uses direct host identifiers instead of hierarchical prefixes to route packets using a greedy source-based policy. When a packet is received, the data plane compares the packet's destination host identifier (ID) with IDs of nodes that are available in the forwarding table and in a database of recently cached source routes (pointer cache). The packet is forwarded to the closest of the matched entries.

In our system, the ROFL data plane stores IDs in sorted order within a TCAM-based forwarding table. We modified the TCAM lookup algorithm to return the shortest ID match instead of the longest prefix match as in IPv4. A second TCAM implemented within the FPGA is used as a routing cache. When packets arrive at the data plane, the forwarding logic extracts the destination host ID from the packet header. The ID is then used for simultaneous searches in the forwarding table and the routing cache. The data plane uses the lowest ID among the search results to forward the packet.

3.3 Software Data Planes

Software data planes provide low throughput extensions to the data planes implemented in the FPGA. Additionally, they usefully enhance the isolation properties of the virtualization platform by forwarding packets during FPGA downtime that would ordinarily be forwarded from hardware data planes. We use container virtualization techniques to implement the software data planes. Container virtualization techniques are popular because of their strong isolation properties and ease of deployment.

We virtualize the Linux server attached to the NetFPGA card using OpenVZ, which virtualizes a physical server at the operating system level. Each virtual machine performs and executes like a stand-alone server. The OpenVZ kernel provides the resource management mechanisms needed to allocate resources such as CPU cycles and disk storage space to the virtual machines. Compared with other virtualization approaches, such as full virtualization and paravirtualization [27], OS-level virtualization provides the best tradeoff between performance, isolation and ease of deployment. The performance difference between a virtual machine in OpenVZ and a standalone server is almost negligible [28]. 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.

The OpenVZ containers run Click as a user-mode program to execute virtual data planes. Click allows data plane features to be easily customized. Although Click offers the best packet forwarding performance when executed in the kernel space [14], we choose to execute Click as a user-mode program by virtue of its ability to be easily deployed and administered.

4 Data Plane Scaling

We consider two separate approaches to scale the number of data planes beyond the logic capacity of the FPGA. In the first approach, all packets initially enter the NetFPGA card. The CPU transceiver module within the FPGA forwards packets targeted for virtual networks implemented in software to the host PC via the PCI bus. Click routers running in OpenVZ containers process the packets and return them back to the NetFPGA card. Processed packets are transmitted through NetFPGA's physical interfaces. We subsequently refer this approach as the **single receiver approach**. In the second **multi-receiver approach**, the NetFPGA card only receives packets targeted for hardware data planes. A separate PC network interface card (Figure 3) receives and transmits packets destined for software virtual data planes. We describe the details of each approach below.

Single-receiver approach: If an incoming packet does not have a match for a hardware virtual data plane in the dynamic design select table on the FPGA, the packet is sent to the CPU transceiver module shown in Figure 3. 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 system exposes CPU DMA queues as virtual Ethernet interfaces to the host OS. The CPU transceiver modules modifies the layer 2 address of the packet to match the address of the virtual Ethernet interfaces of the target software data plane. The kernel software bridge forwards the Ethernet packet to its respective OpenVZ container based on its destination layer 2 address (DST MAC for IPv4 in Figure 4). 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 for the IPv4 data plane). The software bridge then sends the packet to a CPU RX Q on the NetFPGA board via the PCI bus. After input arbitration, the dynamic design select module sends the processed packet to the CPU

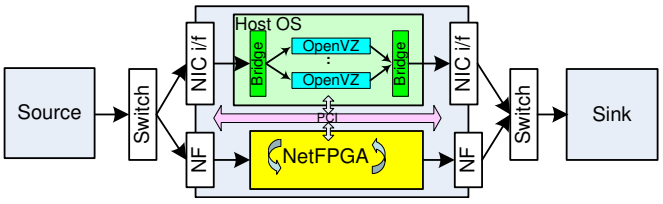

Fig. 5. Multi-receiver setup for scalable virtual networking including dynamic FPGA reconfiguration

transceiver. The CPU transceiver module extracts the source and exit queue information from the processed packet and places it in the output MAC queue interface (MAC TX Q) for transmission.

The software interface enables on-the-fly migration of virtual networks from software to hardware and vice versa. The virtual network operator can dynamically migrate a virtual network from hardware to software in three steps. In the first step, the operator initiates an OpenVZ virtual environment that runs the Click router inside the host operating system. Next, the operator copies all the hardware forwarding table entries to the forwarding table of the host virtual environment. In the final step, the operator writes an entry into the dynamic design select table indicating the association of the virtual IP with a software data plane. Our current implementation imposes certain restrictions on virtual network migration from software to hardware. If the software virtual data plane has a forwarding mechanism that is unavailable in any of the hardware virtual data planes, network migration to hardware requires reconfiguration of the FPGA, as described in Section 4.1.

Multi-receiver approach: In this approach, the NetFPGA card receives packets destined for all FPGA-based data planes while a separate NIC attached to the host PC receives all traffic destined for software data planes. This approach relies on network switches to forward packets to software or hardware data planes, as shown in Figure 5. We use layer 2 addressing to direct each packet to the appropriate destination (NetFPGA card or PC NIC). When deployed in the Internet, we assume 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 (switches) but simplifies the 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.

4.1 Virtual Network Migration

Although virtual networks may be statically assigned to either software or hardware data planes during network allocation, several practical reasons require networks to be dynamically migrated between the two platforms during operation. First, from a service provider's standpoint, the initial virtual network allocation may not be sufficient to support the dynamic QoS requirements of virtual networks during operation. Second, from an infrastructure provider's standpoint, shifting lower-throughput networks to software and higher-throughput networks to hardware can improve the overall utilization of the virtualization platform. Additionally, network migration can reduce the impact of data plane customization on virtual networks in shared hardware. For example, the virtual network operator can migrate unmodified virtual networks in an FPGA to software data planes, reconfigure the FPGA with data plane changes and migrate the networks back to the FPGA to resume operation at full throughput. All unmodified virtual networks can continue their operation at lower throughput using software data planes during FPGA reconfiguration.

We illustrate data plane migration by considering an example where the FPGA is shared by multiple IPv4-based virtual networks for the multi-receiver approach.

1) Before migration, the operator creates Click instances of all active hardware virtual data planes using 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, 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 [26] 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 directed to the PC through the PC NIC (Figure 5) for forwarding with software virtual data planes.
4) The operator now reprograms the FPGA with a new bitstream that incorporates changes in network characteristics. We used a collection of previously-compiled FPGA bitstreams in our implementation.
5) Following FPGA reconfiguration, the operator writes routing tables back to the hardware.
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 for hardware virtual data planes back to the NetFPGA interface. Virtual networks then resume operation in the hardware data planes for the instantiated hardware routers. We quantify the overhead of this dynamic reconfiguration approach in Section 7.

All virtual networks remain fully active in software during the reconfiguration. We use ARP as a mechanism to map virtual IP addresses to virtual MAC addresses. Non-IP data planes can use a similar scheme by incorporating a mechanism to map the non-IP virtual addresses (such as flat labels) to the physical (MAC) addresses.

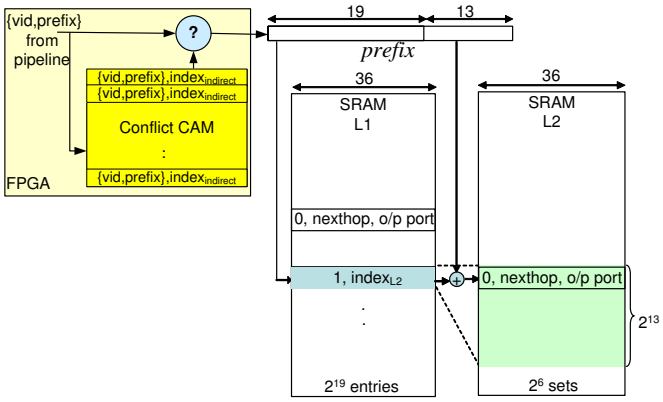

Fig. 6. Architecture of SRAM-based IPv4 forwarding tables

Custom elements written using Click can be used to perform such mapping. ARP remapping is carried out only for those data planes that are executed in hardware. Data planes that are originally executed in software can continue to forward packets in software without requiring ARP address remapping.

Network service requirements and the availability of virtualization resources are also subject to real-time variations. It is therefore important to cleanly separate service requirements from virtualization resources. This separation can be achieved using a scheduling interface that maps service requirements to virtualization resources while maximizing the overall utilization (bandwidth, latency, etc.) of the virtualization platform. Our system implements a simple greedy scheduling technique to assign virtual networks to hardware or software data planes so that the overall bandwidth of the virtualization platform is maximized while aggregate bandwidth and capacity limitations in both platforms are respected. The scheduler attempts to greedily pack low-throughput virtual networks into OpenVZ containers. If a network cannot be executed in a software plane due to bandwidth limitations, it is assigned to a hardware plane. The scheduler recomputes virtual network assignments whenever a virtual network is removed from the platform or when service requirements change during operation. The output of the scheduler can be used by the operator to perform virtual network migrations.

5 Forwarding Table Scaling

Realistic forwarding table implementations that store hundreds of thousands of forwarding entries necessitate the use of inexpensive off-chip memory chips, including SRAMs. However, the design of SRAM-based forwarding tables is not straightforward, particularly for IP-based data planes, because SRAMs lack the parallel search mechanisms that are found in single cycle lookup TCAMs. Additionally, when virtual network operators who share the hardware virtualization platform independently choose prefix address spaces, the overlapped address spaces can lead to contentions in SRAM memory locations (prefix conflicts).

We present the details of external SRAM-based forwarding table architectures for IP (IPv4) and non-IP based (ROFL) data planes below.

5.1 IPv4

The forwarding table architecture for IPv4 (Figure 6) extends the popular DIR-24-8-BASIC technique [21] used for high speed SRAM-based prefix lookups. The DIR-24-8-BASIC technique exploits the bias towards certain prefix lengths in typical backbone routers. For example, 99.93% of IPv4 prefixes have length 24 bits or less [21]. By expanding all prefixes of length 24 bits or less and relocating these prefixes to SRAM locations that can be accessed with single memory access, the average prefix lookup time can be minimized.

Each virtual forwarding table in hardware is identified by a unique identifier (VID). The 36-Mbit SRAM located external to the FPGA is organized as two 18-Mbit memory banks. Each memory bank consists of 2^{19} (512K) entries, where an entry is 36 bits wide. The first bank (L1 in Figure 6) stores all prefixes whose lengths are less than 19 bits. The second bank (L2 in Figure 6) is divided into multiple *sets* with each set consisting of 2^{13} entries. The second bank stores prefixes whose lengths are greater than 19 bits.

Updates: When a virtual prefix of length $l \leq 19$ bits needs to be stored, the control plane software writes 2^{19-l} entries into L1. Each entry is 36-bit wide and consists of a 1-bit flag, 3-bit output port and 32-bit next hop address. The flag bit is set to 0 for prefixes of length $l \leq 19$ bits. For prefixes of length $l > 19$ bits, an entry indexed by the most significant 19 bits of the prefix is written. The flag bit of this entry is set and the remaining bits point to an index location in L2. L2 reserves a set of 2^{13} entries for each prefix of length $l > 19$ bits. Each entry in the set corresponds to one of the longer 2^{13} prefixes indexed by the shared entry in L1. The entries in L2 store the 3-bit output port information followed by the 32-bit next hop entry. This approach can be scaled to cover 99% of all IPv4 prefixes with a 72-MByte SRAM.

The SRAM can be conveniently shared between multiple virtual routers when prefixes do not conflict with each other. However, when virtual prefixes from multiple virtual routers index to one or more exact locations in SRAM, conflicts exist. To resolve each conflict, the control plane software first calculates an indirect index in SRAM to which a conflicting prefix can be relocated. The control plane software then writes the next hop and output port information into the SRAM locations addressed by the indirect index. The indirect index is determined by the control plane on a first-fit basis from the available pool of SRAM locations. The control plane additionally writes the virtual router ID (VID), original prefix and the indirect address into a TCAM implemented in the FPGA (*Conflict CAM*). The Conflict CAM is used to detect prefix overlaps during lookups with a single cycle overhead.

Lookups: The data plane extracts the destination virtual IP address and constructs a *lookup address* by

prepending the virtual IP address with the VID information obtained from the dynamic design select module. The forwarding logic searches for this lookup address in the Conflict CAM. If a match is found, the indirect index obtained from the Conflict CAM is used to index L1. Otherwise, the data plane uses the virtual IP address to directly index into the L1 table. If the most significant bit (MSB) of the L1 table entry is set, the L1 entry is combined with least significant 13 bits of the prefix to obtain an index into L2. Otherwise, the next hop information can be directly obtained from L1.

5.2 ROFL

Each ROFL virtual data plane uses a forwarding table to store ordered resident host IDs and a pointer cache to cache recent source routes. We implement the forwarding table in external SRAM since it is likely to use more memory resources than the pointer cache. The pointer cache is implemented using the TCAM memory within the FPGA. The control plane maps the circular namespace of each virtual router onto a continuous block of SRAM locations. This mapping is achieved by using a hash of the virtual router ID and the namespace base address. Several virtual routing tables can share the SRAM by partitioning the SRAM into multiple namespaces, each belonging to a virtual router. Each SRAM location corresponds to a label in the namespace. The forwarding table only stores a limited set of labels (valid labels). The SRAM locations corresponding to these labels store the egress port information for these labels. All other labels (invalid labels) store the egress port information of the closest label in the namespace.

Updates: To store a new label within a namespace, the control plane software updates the corresponding location in SRAM with the egress port information of the new label. Additionally, the egress port information of all previous invalid labels are set to the new egress port information.

Lookups: The data plane hashes the virtual router ID and destination ID extracted from the packet header into an SRAM location corresponding to the namespace label. Simultaneously, the FPGA forwarding logic searches for the label in the data plane's local pointer cache. The egress port information from the SRAM namespace is compared with the results from the pointer cache and the lowest of the two entries is used to forward the packet.

6 Experimental Approach

To measure the performance of hardware virtual data planes, we use the network configuration shown in Figure 5. Our experiments explore the performance of FPGA-based and OpenVZ-based virtual data planes and the scalability of the integrated system. Performance is determined in terms of network latency and observed throughput. We use the NetFPGA hardware packet generator and packet capture tool [29] to generate packets at line rate and measure network latency and throughput. In general, software data planes cannot handle line rate traffic. We compare the hardware data plane implementations 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.

To measure the scalability of our system, we implemented systems of between 1 and 15 virtual routers. These systems implemented between one and four virtual routers in the FPGA and the rest in host software. While we separately assessed the performance of IPv4 and ROFL data planes, scalability experiments used IPv4 data planes. The Synopsys VCS simulator was used for the functional verification of the hardware designs. All hardware designs were successfully deployed on a NetFPGA cube and executed on the Virtex II FPGA.

7 Experimental Results

We report the performance and resource usage of virtual data planes and analyze the scalability of the virtualization platform in this section.

7.1 Performance

In an initial experiment, we compared the baseline performance of a single hardware virtual data plane running in the NetFPGA hardware and a Click software virtual data plane running in the OpenVZ container. We loaded the packet generator with PCAP files [29] whose packet sizes ranged from 64 to 1024 bytes. These packets were subsequently transmitted to the system at the line rate of 1 Gbps.

We consider four specific system configurations:

1) Hardware data plane with TCAM routing tables - The NetFPGA board receives and transmits all packets. The forwarding tables are stored in a 32 entry TCAM located within the FPGA.
2) Hardware data plane with external SRAM routing tables - The NetFPGA board receives and transmits all packets. The forwarding tables are stored in a 4.5 Mbyte SRAM located external to the FPGA.
3) Click from NIC - The PC NIC (Figure 5) interfaces receive network traffic and use Click data planes executing in OpenVZ containers to forward packets.
4) Click from NetFPGA - The NetFPGA network interfaces receive the traffic. Click data planes forward the packets in OpenVZ containers. The PCI bus transfers packets between the NetFPGA hardware and the OpenVZ container.

7.1.1 Throughput

The throughputs of the four approaches for differing packet sizes are shown in Figure 7. These values show the maximum achievable throughput by each implementation for a packet drop rate of no more than 0.1% of

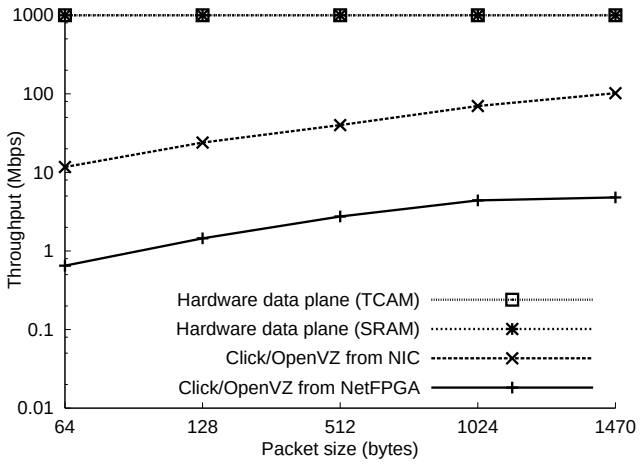

Fig. 7. Receiver throughput versus packet size for a single virtual router

transmitted packets. We measured the receiver throughputs using hardware counters in the NetFPGA PktCap capture tool.

The throughput of shorter packets drops considerably in the software-based implementations. In contrast, the single hardware virtual data plane 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 data planes take advantage of the parallelism and specialization of the FPGA to overcome these issues.

7.1.2 Latency

We use the experimental setup shown in Figure 8 to measure the latency of all four configurations mentioned above. Unlike our previous work that used the *ping* utility for latency measurements [30], the latency experiments described here use the hardware-based NetFPGA packet generator to accurately generate and capture network traffic. While standard software utilities can only measure network latencies on the order of several milliseconds, the NetFPGA packet generator operating at 125 MHz can report latencies with an accuracy of ± 8 ns. In our test setup, we configured ports 0 and 1 of the packet generator in a loopback configuration to provide a baseline measurement while ports 2 and 3 were attached to the experimental virtual router. We simultaneously transmitted two packets of size 64 bytes through ports 0 and 2 and later captured the forwarded packets from ports 1 and 3. The difference in the arrival timestamp values of the two packets indicate the latency of the experimental data plane. We averaged the observed latencies across ten repeats of the experiment.

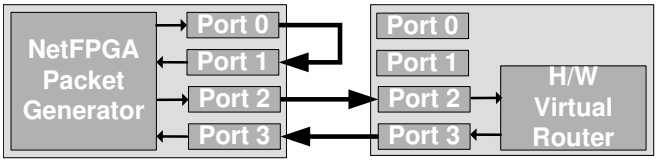

Fig. 8. Experimental setup for measuring latency of SRAM and TCAM forwarding tables

TABLE 1
Data plane latency for IPv4 and ROFL. Both long and short prefixes are used

	IPv4		
Configuration	**Prefix Type**	**Cycles/ Freq(Mhz)**	**Avg. Latency (ms)**
Hardware data plane (TCAM)	**Short/ Long**	1/62.5	3.01
Hardware data plane (SRAM)	**Short**	6/62.5	3.02
	Long	21/62.5	3.17
Click from NIC	**Short/ Long**	-	262.30
Click from NetFPGA	**Short/ Long**	-	408.20
	ROFL		
Hardware data plane (TCAM)	-	1/62.5	2.45
Hardware data plane (SRAM)	-	4/62.5	2.40

Table 1 shows the latency of a single data plane for all four configurations. For SRAM hardware data planes, we separately evaluated the performance of short (length $\leq$ 19 bits) and long prefixes (length $>$ 19 bits) to examine the overhead of two-level memory access required for long prefix lookups in external SRAM. In general, the hardware data planes incur one to two orders of magnitude less latency than software data plane implementations. Although the external SRAM-based forwarding table requires 5 additional cycles for each short prefix lookup than its TCAM counterpart, the observed network latency increases by only 0.1 msec. The moderate increase is justifiable given the large number of prefixes that can be stored in the external SRAM. Longer prefixes incur an additional 15 cycles due two memory accesses, resulting in a 5% increase in the observed latency. The ROFL data plane uses 4 cycles for each lookup.

The additional cycles consumed for SRAM-based IP lookup and ROFL lookup does not necessarily limit the packet forwarding performance. In fact, the impact of higher latency on the overall throughput of the virtualization platform can be hidden by exploiting the pipelined nature of the design. We determined that a 32x32 FIFO buffer inserted between the forwarding logic and the dynamic design select module is sufficient to sustain the line throughput (1 Gbps). The resultant increase in the FPGA logic requirement was less than 1%.

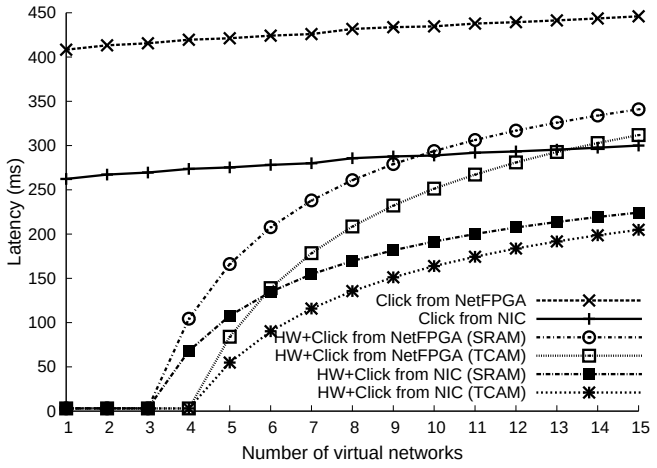

Fig. 9. Average latency for an increasing number of IPv4-based virtual data planes

7.2 Network Scalability

Network scalability can be measured in terms of both throughput and latency. For these experiments, we configured the test topology as shown in Figure 5. Six specific system configurations were considered for systems that consisted of 1 to 15 virtual networks. The software-only Click from NIC and Click from NetFPGA cases are the same as defined in Section 7.1.

Additional cases which combine NetFPGA and software data planes include:

1) Hardware+Click from NIC (SRAM) - The PC NIC receives and transmits all network traffic targeted to OpenVZ-based virtual networks. The NetFPGA physical interfaces receive and transmit all network traffic targeted to FPGA-based virtual networks. This case represents the multiple receiver approach described in Section 4. The hardware virtual data planes use external SRAM-based forwarding tables.
2) Hardware+Click from NIC (TCAM) - This approach is similar to case 1 except that hardware virtual data planes use on-chip-TCAM based forwarding tables.
3) Hardware+Click from NetFPGA (SRAM) - The NetFPGA network interfaces receive and transmit all network traffic. Hardware virtual data planes perform some of the forwarding operations while the rest are handled using Click data planes in OpenVZ containers. 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 4. The hardware virtual routers use external SRAM-based forwarding tables.
4) Hardware+Click from NIC (TCAM) - This approach is similar to case 3 except that hardware virtual data planes use on-chip TCAM based forwarding tables.

For cases 2 and 4, we implemented up to four virtual data planes in the FPGA and the rest (up to 11) as Click processes executing within OpenVZ containers. For cases 1 and 3, we deployed up to three virtual data planes in the FPGA and remaining networks (up to 12) in software. The setup to measure transmission latency for the four cases is shown in Figure 8. As shown in Figure 9, 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 15% from one to fifteen virtual data planes. 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 about 50% additional latency overhead than when they are routed through the NIC interfaces. The average latency of hardware data planes remains constant for up to four data planes. After this, every additional software router increases the average latency by 2%.

To measure aggregate throughput when different numbers of virtual data planes are hosted in our system, we transmitted 64-byte packets with an equal bandwidth allocated for all networks. Next, we incrementally increased the bandwidth share of each virtual network until the networks began to drop more than 0.1% of the assigned traffic. A single OpenVZ software virtual data plane can route packets through the PC NIC interface at a bandwidth up to 11 Mbps. The throughput dropped by 27% when fourteen additional software data planes were added. The software virtual data plane 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 FPGA sustains close to line rate aggregate bandwidths for up to four data planes. The average aggregate bandwidth drops when software data planes are used in addition to FPGA-based data planes.

The top two plots (HW+Click from NIC and HW+ Click from NetFPGA), which overlap in Figure 10, 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.

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

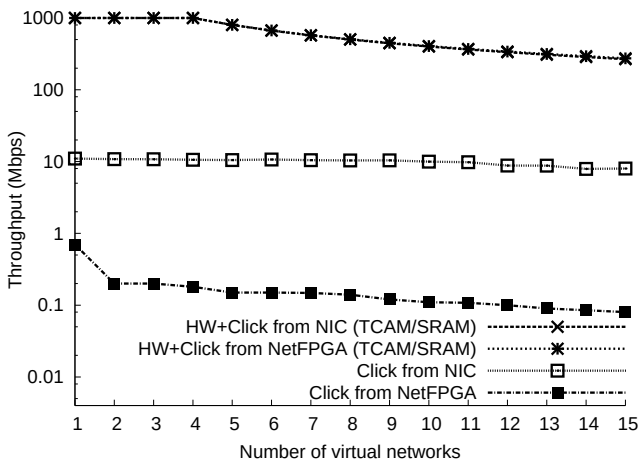

Fig. 10. Average throughput for an increasing number of IPv4-based virtual data planes

7.3 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 data plane. Next, we sent ping packets to the system at various rates which were then forwarded using the NetFPGA hardware plane. Next, we periodically migrated the hardware plane to an OpenVZ container in host software using the procedure described in Section 4.1. After FPGA reconfiguration, we moved the data plane back to the NetFPGA card. We determined that it takes approximately 12 seconds to migrate a hardware data plane 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 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 the maximum sustainable throughput of OpenVZ-based data planes, our system can gracefully migrate the virtual router between hardware and software without any packet loss.

To examine the impact of frequent dynamic reconfiguration on a data plane implemented in an FPGA, we performed an analysis based on experimentally-determined parameters. Consider a situation where a hardware data plane is unchanged for an extended period of time, but must be occasionally migrated from hardware to software when a different hardware data plane is updated or replaced. The overall bandwidth of the unchanged data plane can be represented as:

$$B_{avg} = \frac{B_{sw} * t_{reconfig} + B_{hw} * (T - t_{reconfig})}{T} \quad (1)$$

where B_{hw} represents the aggregate bandwidth of FPGA data planes, B_{sw} represents the aggregate bandwidth of software data planes, $t_{reconfig}$ represents the time required to update the FPGA including FPGA reconfiguration time, and T represents the period of time between FPGA reconfigurations. For our analysis, we assume that four FPGA-based data planes with an individual throughput of 1 Gbps (B_{hw} = 1000 Mbps) are reconfigured in 12 seconds ($t_{reconfig}$ = 12 s), based on our experimentally-collected results. During reconfiguration, all active virtual networks are migrated to host software using the procedure described in Section 4.1 and software data planes offer an aggregate throughput of 11 Mbps with 64 byte packets (B_{sw} = 11 Mbps). Based on (1), if reconfiguration is performed every 15 seconds, the average throughput (B_{avg}) of unchanged hardware datapaths drops from 1 Gbps to 200 Mbps. However, if reconfiguration takes place once every 2 minutes, the average throughput only drops 10% to about 900 Mbps.

TABLE 2
Resource utilization of IPv4 and ROFL data planes

TCAM Lookup						
	ROFL	**IPv4**				
#Planes	**1**	**1**	**2**	**3**	**4**	**5**
Slices	10321	10068	12882	15696	18509	21322
Slice FF	9094	8964	11269	13574	15879	18184
LUTs	14787	15272	19744	24216	28689	33161
IO	437	437	437	437	437	437
BRAM	40	25	40	55	70	85
SRAM Lookup						
	ROFL	**IPv4**				
#Planes	**1**	**1**	**2**	**3**	**4**	**5**
Slices	16146	17867	20030	22202	-	-
Slice FF	11338	12307	13869	15431	-	-
LUTs	24023	26650	30260	34178	-	-
IO	437	437	437	437	-	-
BRAM	10	19	22	28	-	-

TABLE 3
Percentage of prefixes which overlap

BGP Table	Total Prefixes	Prefix Overlap
rrc12	339K	13.0%
rrc13	346K	12.6%
rrc15	339K	15.0%
rrc16	345K	13.8%

7.4 Resource Usage

When internal forwarding tables are used, the Virtex II Pro FPGA can accommodate a maximum of five virtual data planes, each with a 32-entry TCAM-based forwarding table. When the CPU transceiver module is included, the FPGA can accommodate a maximum of four virtual data planes. 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 2 shows the resource utilization of up to five IPv4 virtual data planes and a single ROFL data plane. All designs operate at 62.5 MHz. Synthesis results for the virtual router design implemented on the largest Virtex 5 (5vlx330tff1738) show that a much larger FPGA could support up to 32 IPv4 virtual data planes.

When external SRAM-based forwarding tables are used, the FPGA can only store up to 3 virtual data planes. We attribute the reduction in the number of data planes to the additional overhead of DRAM arbitration logic used for implementing the input and output queues. The DRAM arbitration logic alone consumes about 15% of the overall FPGA resources. A hardware virtual data plane that incorporates the DRAM and SRAM arbitration controllers with a 32-entry Conflict CAM consumes 66% of the total slices and 47% of the total registers. However, we do not expect the logic cost of the arbitration logic to scale with the number of virtual data planes. Larger FPGAs in the Virtex 5 family are able to amortize the additional cost with additional data planes.

7.5 Size of the Conflict CAM

The size of the Conflict CAM is an important design consideration for hardware IPv4 data planes since it uses internal FPGA memory resources to store overlapped prefixes. The size of the Conflict CAM heavily depends on the amount of prefix overlaps between different virtual data planes. Unfortunately, for experimental purposes it is difficult to estimate the amount of prefix overlaps due to the lack of availability of realistic virtual router forwarding tables.

The RIS [31] project provides snapshots of Border Gateway Protocol (BGP) routing tables collected from Internet backbone routers. Although these sample routing tables contain large numbers of prefixes, they do not necessarily represent realistic forwarding tables since the prefixes generally tend to be highly similar across tables. Song, et al. [23] observed that virtual routers in the future Internet are unlikely to have similar prefixes. Existing VPN services, for instance, largely use dissimilar prefixes with different prefix aggregation schemes. Hence, for our analysis, we construct synthetic forwarding tables by partitioning four existing publicly-available BGP routing tables, as shown in Table 3.

We uniformly distributed a set of 100K prefixes chosen randomly from each BGP table between four virtual forwarding tables. Next, we calculated prefix overlaps for each virtual data plane and then averaged them across all four virtual routers. In general, each forwarding table exhibits 12-15% prefix overlap with prefixes found in other tables. Each overlapped prefix in a system with n virtual data planes needs $log(n)$ bits for the virtual ID, 32 bits for the virtual prefix and 19 bits for the indirect index. A system with 4 FPGA-based virtual data planes that stores 100K prefixes with 13% prefix overlap will need approximately 663 Kbits of on-chip memory for the Conflict CAM. The on-chip resources of modern FPGAs, such as those in the Virtex-5 family, are sufficient to address this memory requirement.

8 Conclusions and Future Work

We have presented a heterogeneous network virtualization environment that uses host virtualization techniques to scale existing FPGA-based virtualization platforms. An important contribution of this work is the development of a scalable virtual networking environment that includes both hardware and software data plane implementations. A full suite of architectural techniques are used to support this scalable environment including dynamic FPGA reconfiguration and a forwarding table for the FPGA routers which is optimized for virtual routing. In the future, we plan to evaluate various techniques to improve the programmability of our system.

Acknowledgments

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

References

[1] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet at your spare time," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 1256–1261, Jan. 2007.

[2] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Sep. 2006, pp. 3–14.

[3] 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," in *Proceedings of the ACM Conference on Emerging Network Experiment and Technology*, Dec. 2008, pp. 72–77.

[4] M. Anwer and N. Feamster, "Building a fast, virtualized data plane with programmable hardware," in *Proceedings of the ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures*, Aug. 2009, pp. 1–8.

[5] G. Lu, Y. Shi, C. Guo, and Y. Zhang, "CAFE: a configurable packet forwarding engine for data center networks," in *Proceedings of the ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow*, Aug. 2009, pp. 25–30.

[6] "NetFPGA user's guide," http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide.

[7] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "ROFL: Routing on flat labels," in *Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, Aug. 2006, pp. 363–374.

[8] "OpenVZ project page," http://www.openvz.org/.

[9] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *Proceedings of the 10th International Conference on Passive and Active Network Measurement*, Apr. 2009, pp. 3–12.

[10] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, Aug. 2004, pp. 303–320.

[11] L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron, "Cashmere: resilient anonymous routing," in *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation*, Aug. 2005, pp. 301–314.

[12] "Cisco Nexus 1000V Series Switch," http://www.cisco.com/en/US/products/ps9902/.

[13] J. Turner, P. Crowley, , J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman, and D. Zar, "Supercharging PlanetLab: A high performance, multi-application, overlay network platform," in *Proceedings of the ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2007, pp. 85–96.
[14] E. Keller and E. Green, "Virtualizing the data plane through source code merging," in *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, Aug. 2008, pp. 9–14.
[15] Y. Liao, D. Yin, and L. Gao, "PdP: Parallelizing data plane in virtual network substrate," in *Proceedings of the ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures*, Aug. 2009, pp. 9–18.
[16] ——, "Europa: Efficient user mode packet forwarding in network virtualization," in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking*, Apr. 2010, pp. 6–6.
[17] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Proceedings of the ACM Workshop on Hot Topics in Networks*, Oct. 2009.
[18] N. M. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, Apr. 2010.
[19] M. B. Anwer, M. Motiwala, M. bin Tariq, and N. Feamster, "Switchblade: A platform for rapid deployment of network protocols on programmable hardware," in *Proceedings of the ACM SIGCOMM*, Aug. 2010, pp. 183–194.
[20] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8 –23, Mar. 2001.
[21] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 1998, pp. 1240–1247.
[22] J. Fu and J. Rexford, "Efficient IP-address lookup with a shared forwarding table for multiple virtual routers," in *Proceedings of the ACM CoNEXT Conference*, Dec. 2008, pp. 21:1–21:12.
[23] H. Song, M. Kodialam, F. Hao, and T. Lakshman, "Building scalable virtual routers with Trie braiding," in *Proceedings of IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[24] G. Watson, N. McKeown, and M. Casado, "NetFPGA: A tool for network research and education," in *Proceedings of the Workshop on Achitectural Research Using FPGA*, Feb. 2006, pp. 160–161.
[25] 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," in *Proceedings of the ACM SIGCOMM Conference on Data Communication Applications, Technologies, Architectures, and Protocols for Computer Communication*, Nov. 2008, pp. 231–242.
[26] "Click element documentation," http://read.cs.ucla.edu/click/elements/iproutetable.
[27] VMWare, "Understanding full virtualization, paravirtualization, and hardware assist," Sep. 2007, http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf.
[28] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems*, Mar. 2007, pp. 275–287.
[29] G. A. Covington, G. Gibb, J. W. Lockwood, and N. McKeown, "A packet generator on the NetFPGA platform," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2009, pp. 235–238.
[30] D. Unnikrishnan, R. Vadlamani, Y. Liao, A. Dwaraki, J. Crenne, L. Gao, and R. Tessier, "Scalable network virtualization using FPGAs," in *Proceedings of the ACM SIGDA International Symposium on Field Programmable Gate Arrays*, Feb. 2010, pp. 219–228.
[31] "Routing Information Service (RIS) project," http://www.ripe.net/ris/.

Deepak Unnikrishnan is a Ph.D. candidate in the Electrical and Computer Engineering department at the University of Massachusetts, Amherst. He received the M.S. degree in electrical and computer engineering from UMass in 2009. He was a design engineering intern with Altera Corporation during 2010 and 2012, where he was involved with the design and modeling of embedded high-speed transceiver blocks. His research interests include FPGAs, computer architecture, and virtualization.

Ramakrishna Vadlamani received the B.E. degree in electronics engineering from Veermata Jijabai Technological Institute, Mumbai, India in 2004 and the M.S. degree in electrical and computer engineering from the University of Massachusetts, Amherst in 2010. He is currently with Qualcomm Inc., Boxborough, MA, where he is involved in the design and verification of next generation basestation modems.

Yong Liao graduated with a B.S. degree in 2001 from the University of Science and Technology of China. In 2004, he received the M.S. degree from the Graduate School of Chinese Academy of Sciences. He received his Ph.D. in electrical and computer engineering from the University of Massachusetts, Amherst in 2010. His research interests include inter-domain routing, data center network, and network virtualization.

Jérémie Crenne received his M.S. and Ph.D. degrees from Université de Bretagne Sud, Lorient, France in 2008 and 2011, respectively. Between 2009 and 2010, he was a visiting student at the University of Massachusetts, Amherst. He is currently a postdoctoral fellow at LIRMM, Montpellier, France. His research activities are in the areas of field-programmable architecture and embedded systems security.

Lixin Gao (F'IEEE10) is a professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. She received her Ph.D. degree in computer science from the UMass in 1996. Her research interests include multimedia networking and Internet routing and security. Between May 1999 and January 2000, she was a visiting researcher at AT&T Research Labs and DIMACS. She is an Alfred P. Sloan Fellow and received an NSF CAREER Award in 1999.

Russell Tessier (M'00-SM'07) received the B.S. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1989, and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, in 1992 and 1999, respectively. He is an associate professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. His research interests include computer architecture, FPGAs, and system verification.