Expanding OpenFlow Capabilities with Virtualized Reconfigurable Hardware

Stuart Byma, Naif Tarafdar, Talia Xu, Hadi Bannazadeh
Alberto Leon-Garcia, Paul Chow
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 3G4
{bymastua, tarafda1}@eecg.toronto.edu
{talia.xu, hadi.bannazadeh, alberto.leongarcia}@utoronto.ca, pc@eecg.toronto.edu

ABSTRACT

We present a novel method of using cloud-based virtualized reconfigurable hardware to enhance the functionality of OpenFlow Software-Defined Networks. OpenFlow is a capable and popular SDN implementation, but when users require new or unsupported packet-processing, software processing in the OpenFlow controller cannot provide multi-gigabit rates. Our method sees packet flows redirected through virtualized hardware with custom-designed packet-processing engines that can add new capabilities to an OpenFlow network, while retaining line-rate processing. A case study shows this can be achieved with virtually no loss in throughput and minimal latency overheads.

Categories and Subject Descriptors

B.0 [**Hardware**]: General

Keywords

Cloud Computing, Software-Defined Networking

1. INTRODUCTION

Software-Defined Networking (SDN) is a new networking paradigm that has been gaining popularity in the past few years [1]. The key to SDN is the separation of the network control plane from the data plane, where the control of network nodes is moved to a centralized system defined by software applications. Network nodes become "dumb" programmable switches that are told what to do with incoming packets by the central controller that has full view of the entire network topology. This allows network administrators to write software to achieve extensive fine-grain control over the entire network.

A problem arises when switches do not implement desired features in hardware. This can cause packets to traverse a much slower software path in the switch or through the SDN controller, which usually runs on a commodity server.

In this paper we introduce a novel method for circumventing this slow software path, keeping all desired packet-processing in fast hardware datapaths. We leverage existing work that provides virtualized in-network reconfigurable hardware accelerators, which can be brought up on the fly much like virtual machines [2]. We use these accelerators to implement the desired packet-processing features, and then redirect packet flows through the accelerators to integrate their functionality into the network.

FPGA'15, February 22–24, 2015, Monterey, California, USA.

http://dx.doi.org/10.1145/2684746.2689086.

2. BACKGROUND

In this section we introduce relevant background material on OpenFlow, as well as the cloud-based FPGA infrastructure that we make use of.

2.1 OpenFlow

OpenFlow [3] is a Software-Defined Networking protocol. OpenFlow operates by programming *flows* into OpenFlow-compatible network switches, which match packets against the flows and perform associated actions such as Forward or Drop. Current OpenFlow switches generally support a limited subset of the protocol in hardware. Therefore, a user operating an OpenFlow network at this point in time may come across two roadblocks: 1) An OpenFlow switch does not support a particular optional feature; 2) The user desires a custom rule match or action that OpenFlow does not specify.

2.2 Heterogeneous Cloud Infrastructures

Cloud computing is a paradigm in which physical resources are virtualized so that they can be shared between users in a flexible, scalable manner.

In this paper we will focus on Infrastructure as a Service (IaaS) type clouds. The resources provided in IaaS clouds are typically Virtual Machines (VM), storage, and networking resources, a well-known example being Amazon Web Services. Emerging next generation cloud architectures also include more heterogeneous computing resources such as GPUs, and more recently, FPGAs, as well as virtualized network resources. Figure 1 shows this architecture, components of which are described below.

2.2.1 FPGA Resources (VFRs)

FPGAs typically do not map well to existing cloud systems because there is no straightforward way to abstract and virtualize them. Recent work, however, has used partial reconfiguration to partition a single FPGA device into several reconfigurable regions that are then offered as individual cloud resources [2]. A provided automated compile system allows a user to simply write HDL code and have it run, via the cloud management system, inside one of these reconfigurable regions, which are termed *Virtualized FPGA Resources* or VFRs. From our perspective (the user), we can

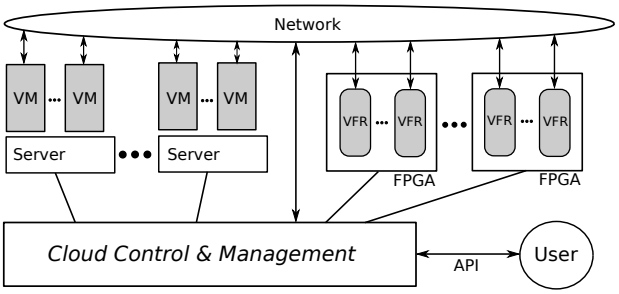

Figure 1: Cloud infrastructure showing VMs and VFRs.

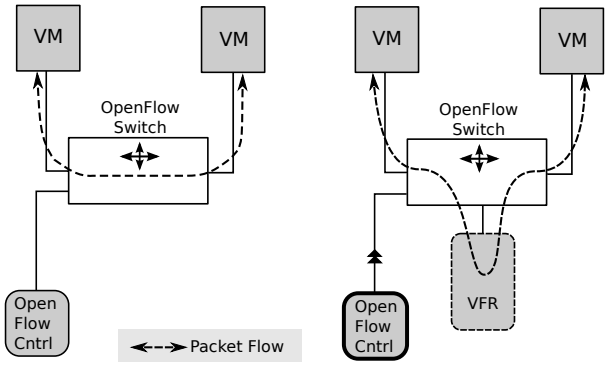

Figure 2: OpenFlow can be used to redirect packet flows through a VFR for additional custom processing.

view VFRs just as we view virtual machines – part of the infrastructure used to implement our cloud-based systems.

2.2.2 Network Resources

Our infrastructure contains a networking system that is also virtualized. This allows us to allocate networks as cloud resources, and attach other resources to these networks as well. These networks are also software-defined, and support the attachment of custom OpenFlow controllers. We make use of the Smart Applications on Virtual Infrastructure (SAVI) network Testbed [4] for our prototyping, which supports this network virtualization, as well as the VFRs discussed above.

3. PROPOSED METHOD

In this section we describe our method of using virtualized reconfigurable hardware to expand the functionality of an OpenFlow network. VFRs are used to implement desired packet-processing features, while *flow redirection* is used to direct packet flows through these VFR-based processors.

3.1 Customizing OpenFlow with VFRs

In a regular, homogeneous IaaS system, customizing an OpenFlow network would invariably involve software packet-processing using virtual machines. However, in our infrastructure, we can use VFRs to accomplish this packet-processing at line rate. The hardware can be booted via the cloud controller, and packet flows can be *redirected* using the programmable SDN that connects all resources.

3.1.1 Flow Redirection

We use the term *Flow Redirection* to describe the process of moving desired packet flows through a VFR. A VFR can be seen as a single port Layer Two network device attached to a specific switch port. Packets need to be sent to the VFR for processing, and packets coming out of the VFR also need to be handled correctly. The scenario is shown in Figure 2. Normally, packets are simply forwarded according to existing flows in the OpenFlow Switch. In flow redirection, packets are redirected to the VFR and processed by custom hardware. Some may be dropped, and some or all may come back out of the VFR's single port. Packets exiting the VFR may also require redirection.

Whatever redirection is required is handled by programming additional flows into the OpenFlow switches, through the user's OpenFlow controller. In general, two sets of flows need to be added – one set redirecting packets into a VFR, and another set routing packets coming out of a VFR.

This flow redirection adds an additional hop to certain paths in the network. This will add latency, however throughput is relatively unaffected, as long as the VFR hardware is designed well. Compared with full network path latencies, which are usually in the millisecond range, the additional latency is comparatively small.

4. APPLICATION CASE STUDY

We present a case study demonstrating our method of enhancing the functionality of OpenFlow networks using reconfigurable hardware, focusing on the domain of network tunnels. We show that additional, even exotic funcionality can be added to an SDN at a low latency cost and negligible throughput overhead.

4.1 VXLAN

The Virtually eXtensible Local Area Network protocol, or VXLAN, is a network tunneling protocol that can connect two local area networks over IP. Entire Ethernet packets are encapsulated inside IP packets, creating a bridge between two physically separate Layer 2 networks. VXLAN is an application layer protocol, running on top of UDP. OpenFlow cannot "see" inside the payload of a UDP packet, meaning that OpenFlow cannot see what kinds of packets are flowing through a VXLAN tunnel. Since VXLAN operates on a designated port, 4789, an OpenFlow network could detect that VXLAN tunnels exist, but it is not capable of anything more than dropping the packets or forwarding them along a certain route. To "see" inside the tunnel requires extending OpenFlow to be able to match and perform actions on new packet fields, which is now possible using flow redirection and virtualized reconfigurable hardware.

4.2 OpenFlow VXLAN Firewall

A custom, OpenFlow-controlled in-network VXLAN Firewall implements the ability to perform matches and actions on the IP addresses, and source and destination ports of an encapsulated packet, and either drop or forward the packet. A user can control forwarding by sending control packets to the firewall hardware to configure it at runtime.

Figure 3 shows a high-level view of the firewall design. The firewall hardware contains two Content Addressable Memory blocks with a depth of 128 each, which store blacklisted IP addresses and port numbers. The hardware parses packets to detect destination UDP port 4789, signifying that the packet is a VXLAN packet. If this port is not detected, the packet is simply dropped. The hardware then tests if the destination transport layer port or the destination IP address of the encapsulated packet matches any currently stored in the CAM blocks. If either the port or the IP address matches, the packet is dropped. Values are written

into the CAMs via control packets. The firewall hardware is limited in complexity due to the area constraints of the current VFRs.

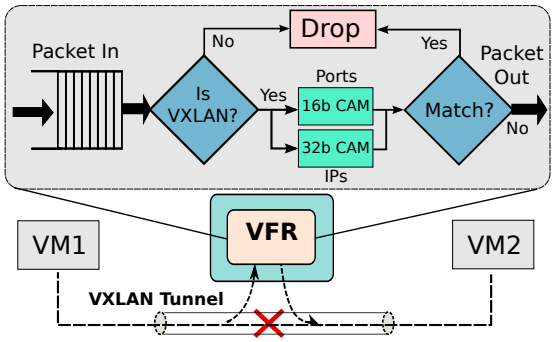

Figure 3: VXLAN Firewall with CAM blocks

4.2.1 *Experimental Results*

We set up an experiment to test the firewall as shown in Figure 2. Two virtual machines are booted on two separate physical machines along with the VXLAN firewall that has been compiled for the virtualized hardware system. Flows are added to the OpenFlow switch connecting these components to perform the redirection of packets traversing the network between the two VMs. The firewall filters the packets according to the rules programmed into it via control packets. Packets forwarded out of the firewall are sent on to their original destination. No modifications need to be made to the VMs; they operate unaware of what is happening in the network. The links of network physical layer are 1G Ethernet. The VMs are Ubuntu machines, with 2GB of RAM and one virtual processor at 1.2 GHz.

The performance utility `iperf` is used to measure throughput from one VM to another, and `ping` is used to measure latency. First, a baseline measurement is taken, that is with no additional flows installed, to see what the throughput and latency are between the VMs without the VFR in the network path. The `iperf` utility is run five times and an average is taken, while `ping` is run until 20 pings are completed. These tests are run both with and without VXLAN tunnelling. Results are shown under No Tunnel and VXLAN Tunnel in Table 1.

Table 1: Throughput and Latency for VXLAN Firewall

	Throughput	Latency
No Tunnel	941 Mb/s	0.465 ms
VXLAN Tunnel	517.4 Mb/s	0.532 ms
VXLAN through VFR	513.2 Mb/s	0.600 ms

Without tunneling the throughput is near line rate (1 Gb/s) as expected. When tunneling using VXLAN, throughput takes a large performance hit due to the software implementation of the tunnel, though the specific loss depends on the VM specs.

Flows are then installed to reroute VXLAN traffic to the running VFR and `iperf` is run again to determine the overhead of rerouting the VXLAN traffic through the VFR firewall. An average of five runs gives a throughput of 513.2 Mb/s, slightly less than the 517.4 Mb/s achieved without the VFR in the network path. This is not a large difference and we can conclude that this technique introduces little to no overhead in terms of throughput. We run the `ping` test again, and results show that rerouting to the VFR introduces a small increase in latency ($\sim$ 12%), but this is expected when adding an additional hop. These results are also summarized in Table 1 under VXLAN through VFR.

4.3 OpenFlow VXLAN Implementation

Our application involves sending a packet between two machines on two different Local Area Networks. The software implementation of encapsulating and decapsulating the packet through a VXLAN tunnel in OpenVSwitch introduces overhead and reduces our line rate throughput of 941 Mb/s to 517.4 Mb/s (Table 1).

We eliminate this software overhead by implementing the encapsulation and decapsulation of packets directly in hardware. The system is shown in Figure 4. Flows are set to forward packets to an *encapsulator* VFR where they enter the tunnel (i.e. encapsulated in a VXLAN packet), after which they are forwarded over the Internet to another VFR, the *decapsulator*, which implements the tunnel egress (i.e. popping the tunneled packet). Control packets are used to set the header information used by the encapsulator, such that the VXLAN packets are destined to the decapsulator. Thus a VXLAN tunnel is created without involving any software packet-processing.

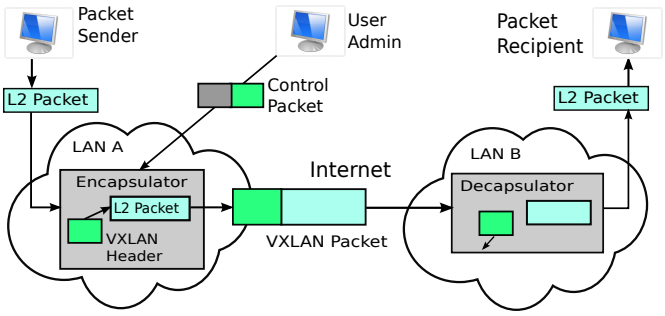

Figure 4: Two hosts on two different LANs communicating using VXLAN.

4.3.1 *Hardware Microarchitecture*

The encapsulator takes an L2 packet and encapsulates it with the entire VXLAN packet header. This packet header consists of 3 portions: MAC header, IP header, and UDP header. The MAC header specifies the source and destination MAC address, where the source is the encapsulator VFR and the destination is the decapsulator VFR. These three packet portions are constant fields that are programmed into the encapsulator with the use of control packets. The encapsulator must first be programmed by the use of three control packets, one specifying each portion of the VXLAN packet header. After the encapsulator has been programmed, it can accept general L2 packets for encapsulation, and entry into the tunnel.

The decapsulator receives an encapsulated packet and pops the original L2 frame/packet. This does not require any additional programming via control packets, since the VXLAN header is a constant size. The hardware does not handle variable IP options fields. The decapsulator checks to see if the packet is a VXLAN packet, and if so, strips off the the header (50 bytes).

4.3.2 *Experimental Results*

The overhead that a software-based tunnel experiences can be quite high, as shown in the first experiment. We

set up another experiment to determine if the VFR-based VXLAN tunnel can indeed alleviate this overhead. We also wish to see how much latency the additional hops in the forwarding path add (through the encapsulator and decapsulator).

We measure the latency and throughput of the round-trip path between the initial sender and final destination, through the encapsulator and decapsulator. All of the nodes are on the same network. We know from the previous firewall application that one redirection added approximately a 12% latency penalty, and therefore we expect to see a similar result here.

The `ping` test is run with the redirection flows installed, and the resulting round trip latency is 0.509 ms, an increase of only 9% from the baseline (non-tunneled) latency of 0.465 ms. This is less than the one-hop redirection in the previous experiment since the latency penalty for the software VXLAN tunnel is not being paid since the tunnel is now implemented entirely in custom hardware. Throughput is nearly unaffected, measured using `iperf` at 938 Mb/s, only a 0.3% difference – verifying that flow redirection with VFRs can nearly eliminate the software overheads of VXLAN tunnels in OVS.

5. RELATED WORK

Related work has looked into using FPGAs with OpenFlow. Some efforts have been made to implement OpenFlow switches on FPGA-based cards – Naous et. al. have used the NetFPGA platform [5] and Khan et. al. have used the NetFPGA10G platform [6]. Their goal was to enable experimental exploration of SDN control and data planes, whereas the work we present here uses available cloud-based FPGAs to enhance existing OpenFlow networks with custom functions.

Much work has been done in the area of utilizing FPGAs to implement custom high-speed packet-processing hardware as well. Packet classification [7, 8], flow matching [9], and deep packet inspection [10] are just a few examples. These approaches may map well into our system to provide these specialized capabilities and algorithms to an existing OpenFlow network.

6. FUTURE WORK

One problem this type of system may face is that of usability and adoption – many potential end-users may not consider it given that it involves complex hardware design. The key may be to provide a high-level specification or domain specific language for describing rules, matches and actions to execute on packets. Recent work and developments in this area could be leveraged to provide a friendly, convenient way to create hardware engines that will work well with flow redirection [11, 12]. Libraries of different VFR compatible hardware engines could be created for use in a cloud system as well.

There are other applications of our method as well – line-rate deep packet inspection could be carried out on flows, with the hardware tailored to detect whatever the user is looking for. Virtualized hardware could realize the routing and forwarding protocols for an overlay network, avoiding the need to implement these nodes on Virtual Machines, thereby likely enhancing system performance.

7. CONCLUSION

In our view, redirecting packet flows through custom in-network hardware presents a compelling case. We have shown that additional, even exotic, functionality can be added to an OpenFlow Software-Defined Network with little to no penalty in throughput, in exchange for a small increase in end to end latency due to the additional hop that flow redirection adds. Recent developments in Domain Specific Languages and high-level synthesis will also make this approach to specialized functionality more palatable to the many potential users who do not have a background in digital hardware design.

8. ACKNOWLEDGEMENTS

We would like to thank members of the SAVI testbed Thomas Lin and Eric Lin for their help in running our experiments. This work was also supported in part by the SAVI Network, NSERC and Xilinx Inc.

9. REFERENCES

[1] Nick McKeown. Software-Defined Networking. *INFOCOM Keynote Talk*, 2009.

[2] Stuart Byma, J Gregory Steffan, Hadi Bannazadeh, Alberto Leon-Garcia, and Paul Chow. FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack. In *Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2014.

[3] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[4] J.M. Kang, H. Bannazadeh, and A. Leon-Garcia. SAVI Testbed: Control and Management of Converged Virtual ICT Resources. In *International Symposium on Integrated Network Management*, pages 664–667. IEEE, 2013.

[5] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an OpenFlow Switch on the NetFPGA Platform. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9. ACM, 2008.

[6] A Khan and N. Dave. Enabling Hardware Exploration in Software-Defined Networking: A Flexible, Portable OpenFlow Switch. In *Field-Programmable Custom Computing Machines (FCCM)*, pages 145–148, April 2013.

[7] J. Fong, Xiang Wang, Yaxuan Qi, Jun Li, and Weirong Jiang. ParaSplit: A Scalable Architecture on FPGA for Terabit Packet Classification. In *High-Performance Interconnects (HOTI)*, pages 1–8, Aug 2012.

[8] Weirong Jiang and V.K. Prasanna. Scalable Packet Classification on FPGA. *IEEE Transactions on Very Large Scale Integration Systems*, 20(9):1668–1680, Sept 2012.

[9] Weirong Jiang, V.K. Prasanna, and N. Yamagaki. Decision Forest: A Scalable Architecture for Flexible Flow Matching on FPGA. In *Field Programmable Logic and Applications (FPL)*, pages 394–399, Aug 2010.

[10] Sarang Dharmapurikar, Praveen Krishnamurthy, T. Sproull, and J. Lockwood. Deep Packet Inspection Using Parallel Bloom Filters. In *High Performance Interconnects*, pages 44–51, Aug 2003.

[11] G. Brebner and Weirong Jiang. High-Speed Packet Processing using Reconfigurable Computing. *IEEE Micro*, 34(1):8–18, Jan 2014.

[12] Xilinx Inc. Xilinx SDNet. `http://www.xilinx.com/applications/wired-communications/sdnet.html`, 2014.