Enabling Network Function Virtualization over Heterogeneous Resources

Thomas Lin, Naif Tarafdar, Byungchul Park, Paul Chow, and Alberto Leon-Garcia
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto, ON, Canada
t.lin@mail.utoronto.ca, naif.tarafdar@mail.utoronto.ca, byungchul.park@utoronto.ca
pc@eecg.toronto.edu, alberto.leongarcia@utoronto.ca

***Abstract*—The economies of scale afforded by cloud computing has been a driving force behind the rapid development and deployment of new cloud-based network applications and services. With the massive growth of IoT devices, we expect a sharp rise in the volume of traffic seen going to and coming from cloud datacenters, which will continue to grow over the next several years. Network Function Virtualization (NFV) is a recent concept which promises to grant network operators the required flexibility to quickly develop and provision new network functions and services in the cloud. As NFV is agnostic to the computing resource, we foresee scenarios where unconventional resources such as FPGAs and GPUs will be of benefit. To this end, we present an architecture based on Software-Defined Infrastructure (SDI) which offers an abstracted control and management interface over virtualized heterogeneous resources in the cloud. Through a unified set of APIs, this architecture enables both application developers and network operators to dynamically deploy and manage new services in the cloud alongside the underlying network that interconnects them, all in a fully software-defined manner. We demonstrate and evaluate an implementation of our NFV-enablement architecture using the SAVI testbed, a multi-tier and SDN-enabled cloud containing virtualized heterogeneous compute resources.**

I. Introduction and Motivation

Cloud computing has been a driving force in the rapid development and deployment of new applications. In the modern era of mobile IoT, a flurry of new applications has resulted in an explosive growth of global traffic emanating from cloud datacenters. It is estimated that by 2020, 92% of workloads will be cloud-based, with the largest contributing applications being video streaming and IoT-related data analytics [1]. We foresee that certain applications and services may require low latency or local storage and processing of sensitive data. Thus, recent works have proposed distributing small-scale cloud datacenters at the edge, or even on premise, closer to the end-users [2][3][4][5]. This in turn has driven a revolution in the operations and management of communication networks, realized in the embrace of the "software-ization" of things.

The recent activities in Software-Defined Networking (SDN) has been a key driver in the evolution of network operations and management. Meanwhile, the rise of Network Function Virtualization (NFV) [6] promises to drive innovation for service providers who embrace the cloud. With NFV, network operators can create virtualized network functions (VNF) using programmable resources in the cloud (e.g. software running in containers or virtual machines). SDN highly complements the recent work on NFV, where network operators or administrators can dynamically spawn VNFs and strategically place them close to where they are needed, while leveraging SDN to direct traffic to the VNFs' location. The benefit of SDN with NFV becomes more apparent when we consider cases of dynamically scaling and migrating existing VNFs, as well as service chaining a series of VNFs to realize a more sophisticated network function. An email service chain, for example, could include anti-virus, spam filtering, and phishing detection. Each component can be implemented as a VNF and SDN can be utilized to dynamically route the email traffic through them.

In the modern era, cloud providers face a large diversity of application and data processing requirements. At a high level, data processing tasks can be categorized as either compute intensive and/or network intensive [7]. Compute intensive tasks require significant CPU resources, while network intensive tasks require high network bandwidths to process high-volume traffic. This begs the question: is there a one-size-fits-all approach to realizing VNFs? Although software-based VNFs provides flexibility and rapid development cycles, they suffer from lower performance compared to dedicated hardware middle boxes when processing compute or network intensive tasks. As the volume and variability of cloud traffic increases, we foresee more demanding network functions that will require higher performance and scalability. In such cases, VNFs will require hardware acceleration based on specialized resources including programmable hardware (FPGAs) and general purpose graphics processing units (GPUs) to meet their performance goals.

To this end, clouds comprising heterogeneous compute resources have been studied and implemented in both academic (e.g. the SAVI testbed [8]) and commercial settings (EC2 [9] and Azure [10]). In each of these clouds, users can easily obtain compute resources, but they must configure and program the resources themselves. Thus, providing specialized heterogeneous resources may be wasteful if there are not enough users with the knowledge or skills to operate them. We argue that the process of providing VNFs should be simpler, where the service provider offers a library of pre-built VNFs for the user to choose from. In a richly heterogeneous infrastructure, the challenge arises regarding how to provision,

manage, and control such diverse resources alongside the network that chains them together in a tightly integrated fashion, and transparently expose their functionality to users. From the users' perspective, it does not matter how VNFs are provided or delivered as long as their desired level of performance is met. Meanwhile, cloud service providers need a method for seamlessly switching to using hardware acceleration if the performance requested by the user is not possible to achieve on generic CPU-based resources (e.g. VMs or containers).

In this paper, we will present the design and implementation of an infrastructure control and management system that enables the creation of heterogeneous NFV service chains in a unified manner. Afterwards, we present an evaluation which motivates the need for hardware-accelerated VNFs, and showcasing our system's ability to hot-plug VNFs on-the-fly. Section II will present the background and related work. Section III will discuss the requirements needed for our heterogeneous NFV enablement system, and is followed by our design and implementation presented in section IV. A functional evaluation in section V will showcase the validity of our system, and section VI will conclude this paper.

II. Background and Related Work

In this section, we provide a brief discussion of background and related works upon which this paper builds on.

A. OpenStack

OpenStack[1] is an open-source cloud computing platform composed of many different sub-projects and services. In this work, the main services we use are Nova and Neutron. Nova is OpenStack's compute virtualization service, and traditionally was only used to virtualize CPUs to create VMs. Nova's design offloads the actual virtualization tasks to pluggable agent processes running within remote servers. We also leverage Neutron, OpenStack's networking service. Similar to Nova's design, Neutron can work with many network management technologies by nature of its pluggable architecture. By itself, Neutron merely stores network state information, and thus relies on vendor plugins to realize the state stored in its database and configure the relevant networking devices (e.g. switches and routers).

B. Smart Applications on Virtualized Infrastructure

As part of the Canadian Smart Applications on Virtualized Infrastructure (SAVI) project[2], we have developed a country-wide multi-tier heterogeneous cloud testbed [8]. The multi-tier aspect refers to the fact that the testbed is comprised of a few Core datacentres, and several "Smart Edge" datacentres. The Core datacentres contain a plethora of CPUs for conventional VMs. The Smart Edge datacentres, which are smaller in size, are physically located closer to end-users and contain various types of heterogeneous resources. This strategic placement of the heterogeneous resources enable applications that require specialized low-latency processing.

One of the main goals of the SAVI testbed, which currently spans eight university institutions, has been to explore the idea that all components of the physical infrastructure can be shared and virtualized. The testbed contains various heterogeneous resources including, but not limited to, a large number of conventional VMs, GPUs, FPGAs, SDRs, IoT sensors, and etc. Research regarding how to best virtualize and share these types of resources in a unified fashion is currently ongoing. While the various datacentres in the SAVI testbed are interconnected via a dedicated 1 Gigabit Ethernet (GE) WAN, within many of the datacentres, the resources are interconnected via high performance 10 and 100 GE OpenFlow-enabled switches. This makes the SAVI testbed the ideal platform upon which to conduct our work.

C. Hardware Acceleration in NFV

ETSI has previously released an NFV Infrastructure specification [11] which presents the high-level architectural pieces required for an infrastructure to natively supports NFV. Following this specification, in [7], Bronstein et al. demonstrated a proof of concept showcasing the benefits of hardware acceleration for certain network and compute intensive VNFs (load balancing and IPsec).

Our work goes further by presenting a framework for enabling hardware VNFs using a unified set of APIs. In addition, we present an architecture that exposes APIs for users to not only acquire their own hardware-accelerated VNFs, but also to dynamically control the network to create unique service chains.

III. Requirements

In this section, we describe the functional requirements and objectives a cloud environment needs in order to support NFV and service chaining over heterogeneous resources.

A. Required Functionalities

To realize NFV service chaining over heterogeneous resources, at minimal there are two things that are required: 1) An infrastructure control and management system with a way to abstract multiple resource types; and 2) A method to control the network flows in order to form new chains or alter existing ones. We elaborate both of these points below.

Abstraction over multiple resources types: An abstraction hides the details of an underlying entity. Although the abstractions provided by a cloud varies based on the cloud computing models (*X* as Service), the most fundamental one is cloud resource abstraction in IaaS. This enables the creation, termination, and configuration of resources. In a traditional cloud, only conventional resources such as compute, memory, and storage were abstracted. However, to realize NFV over heterogeneous resources, providing an abstraction and control over all types of resources through a common set of APIs is required.

Dynamic service chaining: A service chain consists of a set of network functions that are interconnected through the network to support an application. When NFV is deployed in an SDN-enabled network environment, it is possible to dynamically create a chain of virtualized network functions (VNFs). When hardware acceleration of a network function is needed, a user or network operator should be able to easily re-route traffic to

[1]https://www.openstack.org

[2]https://www.savinetwork.ca

hardware VNFs. Dynamic service chaining also enables the sharing of VNFs between different applications, which can reduce operational costs and optimizes the use of available resources.

B. Objectives

While the two aforementioned requirements are essential for our goals, there are objective features which are heavily desirable for improving the system and would help operators and users deploy new NFV-based applications more easily.

Unified APIs: As new resource types are integrated into a given infrastructure, we would like to avoid creating new APIs for each type. Thus, we desire to have a unified API for all types of resources in a given class (e.g. an API for compute resources may include VMs, FPGAs, GPUs; while wireless resources may include SDRs, Wi-Fi APs, and cellular). This keeps the number of APIs limited and easily centralized, and the specific type of resource can be a simple parameter.

Orchestration: Orchestration is the end-to-end automated deployment of services in a cloud environment. It helps to accelerate the delivery of IT services by allowing users to easily define and create new applications, systems, middlewares, and services. An orchestration service should allocate and configure a set of resources as well as configure the network based on user-specified orchestration templates.

Auto-scaling: Auto-scaling allows users to automatically scale their resources according to user-defined conditions, helping them maintain application uptime and performance levels. It is desirable for the NFV enablement service to provide auto-scaling of the VNFs in the service chains. For example, an VNF can be dynamically scaled up (e.g. increasing total compute capacity) when its utilization nears saturation, or scaled down when they are under-utilized.

Monitoring Service: Monitoring of resources is essential for the operation and management of infrastructures, services, and applications. Cloud infrastructures should be able to monitor the status of resources such as the network and CPU utilization, memory consumption, service up-time, and many other performance metrics. Such monitoring features is a very a essential step towards realizing efficient auto-scaling, orchestration, and quick troubleshooting.

VNF Repository: Current clouds do not provide VNFs directly to users, but instead provide the resources needed to create their own custom VNFs from scratch. However, not all users have the skills or knowledge to implement these appliances. Instead, users may wish to deploy pre-built VNFs, or upload their own custom VNFs to be shared with others. The VNF repository keeps pre-built VNF images (e.g., DPI, firewall, load balancer, and etc.) which users can deploy with minimum efforts.

IV. Infrastructure Design and Implementation

We now present a design for realizing a cloud infrastructure management system that meets the functional requirements and objectives discussed in the previous section.

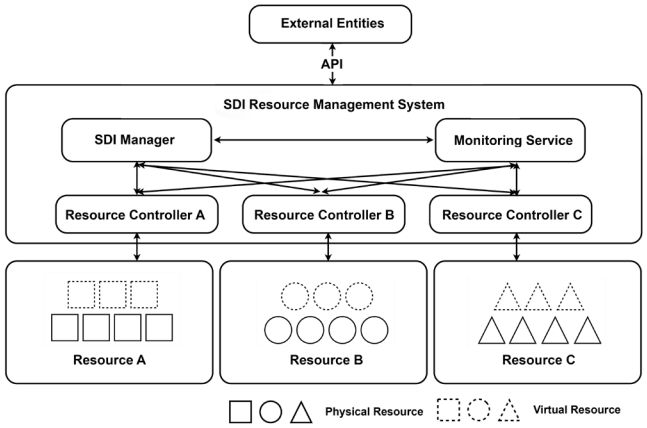

Fig. 1. High-Level Architecture of Software-Defined Infrastructure (SDI)

A. Software-Defined Infrastructure

We utilize a control and management architecture based on Software-Defined Infrastructure (SDI) [2] to orchestrate multiple heterogeneous resources. Similar to how SDN controllers provide interfaces for novel network control applications to be built upon, the "software-defined" aspect of the SDI management architecture, as shown in Fig. 1, refers to the fact that the system provides a set of open programmatic interfaces (APIs) for the enablement of infrastructure control applications. Our in-house SDI management system exposes the necessary APIs to make the acquisition, configuration, management, and retirement of resources a fully software-defined process. While users can create applications using these low-level basic APIs, these applications themselves may in turn provide more complex services by taking high-level requests from users and translating the requests into the lower-level API calls, thus extending the basic capabilities provided by the SDI manager. In this way, users and administrators alike can construct unique PaaS services based on SDI APIs.

Also shown in Fig. 1 is how an SDI manager can liaise with and control multiple resource types. For each resource type within the infrastructure (e.g. x86 compute servers, FPGAs, GPUs, IoT sensors, network switches, access points, etc.), there is an associated controller responsible for interfacing with and configuring those resources. Each resource controller is then interfaced with the SDI manager and associated monitoring service. The set of resource controllers represent an abstraction between the infrastructure resources themselves and the SDI manager, thus enabling the SDI manager to operate while remaining agnostic of the underlying technology. Future resources types can be incorporated into the management framework by interfacing an appropriate virtualization and control system with the SDI manager. This also enables the resource controllers to evolve while keeping the SDI APIs consistent.

B. Heterogeneous Service Chains using SDI

In this section, we discuss how we leverage the SDI architecture to implement a platform which enables flexible instantiation and chaining of heterogeneous VNFs. Figure 2

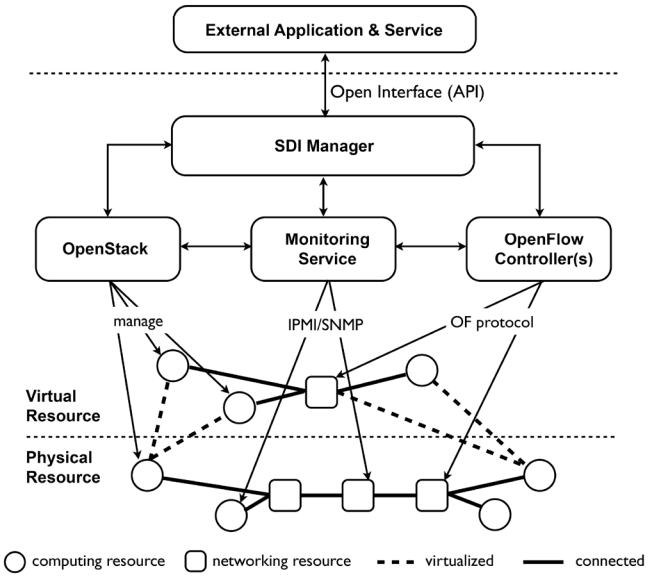

Fig. 2. SAVI Control and Management System based on SDI Architecture

describes the design of a SAVI "Smart Edge" node based on the SDI architecture. The SDI manager leverages OpenStack, an industry standard cloud computing platform, alongside OpenFlow, the de facto standard SDN protocol, to abstract the available resources. The OpenStack and OpenFlow controllers enable direct virutalization and control of the computing and networking resources, respectively. The SDI manager liaises with the controllers to perform converged control and management tasks. We briefly elaborate on each of the vital components and how they meet our requirements.

OpenStack Support: We leverage OpenStack's Nova and Neutron components to help virtualize the compute resources and store network state information. We designed special plugins for both Nova and Neutron for them to inter-operate with our SDI manager, providing it with the necessary information to have an up-to-date view of the infrastructure. In turn, the SDI manager can call the Nova APIs for virtualizing compute resources. By default, OpenStack does not support the virtualization of unconventional resources such as FPGAs or GPUs. Thus, we have extended it to support the virtualization of such resources by modifying OpenStack Nova, which runs in remote physical servers hosting FPGAs and GPUs.

When virtualizing GPUs, we are currently able to offer complete control of the device in two different ways: complete baremetal or as part of a VM. A complete baremetal offering provides the entire physical server within which the GPU resides. This is useful for applications with intensive processing demands that require both the GPU and a large complement of CPUs. Alternatively, we can provision a VM with access to an underlying GPU via PCIe passthrough, a method for passing control of physical PCIe-connected devices through a hypervisor. This may be more desirable for applications that simply pass incoming data directly to the GPU for processing, without need to do much local processing on the CPU.

For virtualizing FPGAs, we have several potential options: complete control over an entire FPGA board, control over a small sub-region of an FPGA board [12], or control over a cluster of several FPGAs [13]. The decision of which one to use depends on a given application's needs. Control over entire boards (multi or single) can be achieved in much the same way as the GPUs, through either complete baremetal or PCIe passthrough. Each of these options are exposed as flavors in our extended version of the OpenStack Nova service. In this way, we are able to support the virtualization of multiple types of compute resources in a unified manner via the standard Nova API.

By default, the ports on an FPGA lacks any specific MAC or IP address information. Our system must assign this information in order for other end-hosts and devices within the network to communicate with it. When a user acquires a new FPGA resource, a new IP and MAC is generated by creating a virtual port via the OpenStack Neutron service. The virtual port is then mapped to the physical switch interface where the FPGA's port is connected by registering it with the SDI manager, thus providing it the information required to direct packets (via OpenFlow control of the network) to and from the FPGA.

OpenFlow Support: In our SDI-based management architecture, the OpenFlow controllers acts as proxies on behalf of the SDI manager, receiving the networking related events from the OpenFlow-enabled switches, and sets up flow table rules and actions according to decisions made by the SDI manager. Such events include not only PacketIn events, but also events related to changes in the topology (e.g. ports going up or down). Such a design allows the SDI manager, with its global view over the state of the entire infrastructure and its resources, to set up informed and optimal network flows according to any SLAs required by the users and their applications.

As we consider the network as just another resources that users should be able to programmatically control, the SDI manager exposes APIs to allow users to install custom networking flow rules into the switches of the infrastructure at layer 2 Ethernet. Normally, the exposure of capabilities at such a low level would open up the system to potential abuse. However, being positioned atop of both OpenStack and the OpenFlow controllers, the SDI manager is well suited to handle the necessary authorization and authentication of incoming flow installation requests from users. We have developed and introduced a conflict avoidance mechanism that ensures different tenants' custom network flows do not interfere with one another, constraining their operation within their authorized realm [14].

Using these basic flow-based networking APIs, we created a higher-level service specifically for enabling dynamic service function chaining. Given two endpoints and a set of VNF middleboxes, this chaining service can adjust the network flow configurations to redirect traffic (bi-directional or uni-directional) to go through the VNF. These flows can also be adjusted dynamically to allow users to swap different VNFs in and out of the service chain, which is particularly useful when needing to scale to a larger virtual appliance or switching from a CPU-based appliance to an FPGA-based implementation.

C. Meeting Objectives

We now consider how we are able to meet the various objectives previously discussed in section III-B.

VNF Repository: To realize a VNF repository, we use the image repository service from OpenStack (Glance) which allows us to upload VM images or take snapshots of existing VMs. Using such a service as a repository for VM-based VNFs is straight forward; however, we can also leverage the same service in order to store raw hardware bitstreams for FPGA-based implementations. A bitstream describes the configuration of the logic on an FPGA to create the desired custom hardware logic. In the case of FPGAs connected to a VM via PCIe passthrough, customized disk images can be created that re-programs the FPGA upon VM bootup. For further customization of the VNFs, the images can contain a server process designed to receive API calls to further configure the FPGA VNF. For GPU-based VNFs, a similar strategy is also applicable by storing pre-built CUDA kernels within bootable disk images using the existing repository service.

Orchestration: To achieve basic orchestration, we utilize the OpenStack orchestration service Heat, which allows us to allocate and configure multiple resources in one shot. Since our system design and implementation allows us to provision heterogeneous compute resources through a unified APIs, we can re-use Heat for these other types of resources. Heat allows us to write plugins to extend the resources it is able to orchestrate, allowing us to integrate it with the SDI manager's exposed set of APIs.

Monitoring & Auto-Scaling: Our SDI-based management architecture has a monitoring co-service called MonArch [15] that is able to liaise with the resource controllers of the the infrastructure to collect data from various sources. The SDI manager is able to leverage this service to query various metrics related to the resources of a given application, and perform big-data analytics to evaluate trends and perform diagnostics. This insight can then be used by the SDI manager to auto-scale resources as needed to meet any pre-programmed SLAs required by applications.

Incremental Heterogeneous Design Flow: The heterogeneity of compute resources enables an incremental design flow for developing FPGA and GPU VNFs. As the development cycle on these devices may be longer than in software, we can first implement a service chain quickly in software and incrementally swap parts of the chain for FPGA or GPU VNFs as they become available, depending on the computation requirements. This also allows cloud service providers to create multiple implementations of the same VNF (e.g. in software and on an FPGA) and allows users to choose VNFs based on their performance requirements.

V. Functional Evaluation

We present two experiments to evaluate the functional correctness and performance of our system.

A. Experiment 1: Hardware-Accelerated Signature Matching

In the first experiment, we aim to motivate and establish the need for hardware-accelerated VNFs for compute-intensive tasks in high-bandwidth settings. Our sample NFV service chain implements a white-list signature filter, where all signatures must be found in order for the packet to pass through. This involves scanning packet payloads for strings (i.e. a simple deep packet inspection).

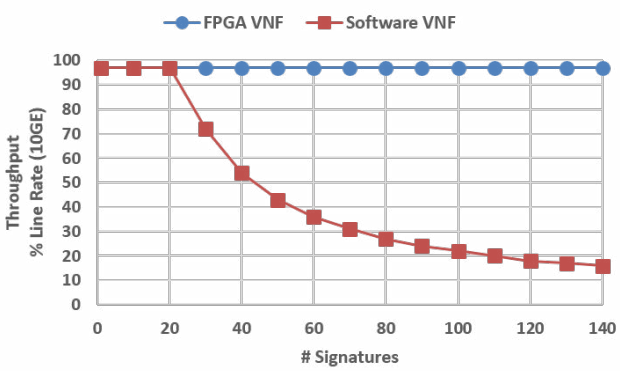

Fig. 3. Throughput of FPGA vs Software Based VNFs

We developed two versions of the string searching VNF, one in software and one using an FPGA. Ensuring peak performance in the software implementation required us to perform many micro-optimizations of the code. We also leveraged the Data Plane Development Kit (DPDK) [16], a library which gives software applications direct kernel access for enabling high-speed packet processing. Additionally, we forced our VM onto a custom server with a CPU clocked at 3.5 GHz, which we note is much higher than the clock speeds on CPU models offered by most public clouds [17][18]. Our FPGA VNF was implemented using Vivado's High-Level Synthesis [19] (HLS), a set of libraries which allows users to write algorithms in C/C++ that gets compiled down to a digital hardware implementation.

We steer a 10 Gbps flow of traffic through the VNF and compare the throughput of the chain with the software-based VNF versus the FPGA VNF. Both implementations utilized the same word search algorithm and processed the same traffic. Our target signatures were fixed to be 10 Bytes in length, and our payloads were 1470 Bytes. We then varied the number of signatures from 10 to 140 signatures. As shown in Fig. 3, the software-based VNF becomes a bottleneck as the throughput dwindles once the number of signatures surpasses 20, dropping significantly afterwards. Meanwhile, the FPGA-based VNF is able to process the traffic at line rate and does not represent a bottleneck in the chain.

As datacentre network traffic continues to grow, we foresee VM-based software VNFs will be unable to cope with compute and network intensive workloads and applications. Such VNF applications may include stateful deep packet inspection, high-bandwidth flow statistics measurements (i.e. without sampling), video processing, traffic encryption/decryption, and so forth.

B. Experiment 2: Application Downtime of Hot-Swapping

In our second experiment, we show the application downtime incurred by our system when hot-swapping a software-based VNF out of an existing service chain and replacing it

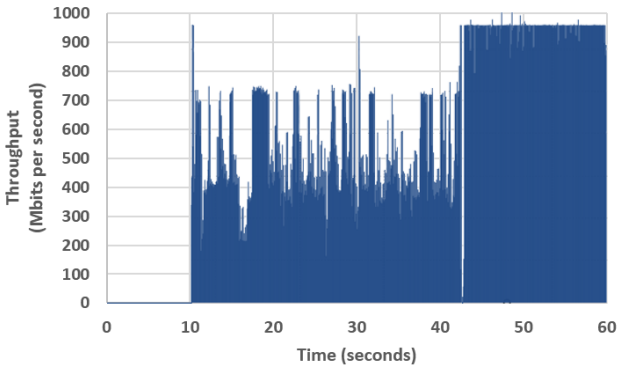

Fig. 4. Throughput During Hot-Swap of Software VNF for Hardware VNF: Downtime of 0.12 second at around 43 second mark

with an FPGA-based VNF. Since the goal here is to simply observe the downtime of the traffic, the VNF is a plain traffic forwarder and no optimization work was done. In this experiment, we utilize a simple UDP flooder as the traffic source. This experiment was conducted in a region of our testbed with only 1 GE links.

Figure 4 shows the results of our experiment. The chain is dynamically formed using the software VNF at roughly 10 seconds in. After about 30 seconds, we hot-swap an FPGA-based VNF in, which involves installing new flows into the network to re-direct the traffic through the FPGA VNF and tearing down the old flows. As seen in the figure, this results in a brief application downtime of 0.12 seconds at around 43 seconds in, during which time packets are dropped. Additionally, it can be seen that the software implementation had trouble keeping up with the traffic source when faced with large numbers of small packets, while the FPGA-based version had no such issues. From these results and our experience, we find that achieving peak performance in software requires much micro-optimization work from the users, while creating hardware implementations via HLS achieves more performance for the same amount of effort.

VI. Conclusions and Future Work

Driven by the surge of IoT and data-related workloads, the migration of existing applications and services to the cloud is expected to accelerate over the next several years. This rapid shift towards the cloud will result in an accelerated growth of traffic volume within datacentres, where we foresee compute and network intensive workloads performed using traditional VM-based software VNFs will run into performance difficulties. This motivates new research on how to best incorporate specialized resources such as FPGAs and GPUs into future cloud infrastructures.

In this paper, an SDI-based architecture for enabling NFV service chains in heterogeneous cloud infrastructures has been presented. Via a unified set of APIs, our architecture performs converged control and management over resources comprising conventional compute and network resources as well as unconventional hardware resources such as FPGAs and GPUs. We designed and implemented the proposed architecture on the SAVI testbed, a multi-tiered cloud infrastructure containing heterogeneous compute, wireless, and IoT resources. We demonstrated how our system enables users to construct unique high-performance network services based on SDN-enabled multi-resource NFVs.

In future work, we will work to populate our library of VNFs for the various resource types within the SAVI testbed. In addition, we will also investigate ways to minimize the application downtime due to hot-swapping VNFs. We also plan to improving our monitoring of services and VNFs such that we can proactively scale them before performance is degraded.

References

[1] Cisco, "Cisco Global Cloud Index: Forecast and Methodology 2015-2020," in *Cisco White Paper*, 2016.

[2] J.-M. Kang, H. Bannazadeh, H. Rahimi, T. Lin, M. Faraji, and A. Leon-Garcia, "Software-Defined Infrastructure and the Future Central Office," in *Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 225–229, June 2013.

[3] IDC, "The Business Benefits of Implementing NFV: New Virtualized vCPE Enterprise Services," Nov. 2014. [Online; accessed 8-June-2015]. Available: http://www8.hp.com/h20195/V2/getpdf.aspx/4AA5-6949ENW.pdf.

[4] L. Velasco, L. Contreras, G. Ferraris, A. Stavdas, F. Cugini, M. Wiegand, and J. Fernandez-Palacios, "A Service-Oriented Hybrid Access Network and Clouds Architecture," *Communications Magazine, IEEE*, vol. 53, pp. 159–165, April 2015.

[5] ETSI, "Mobile-Edge Computing - Introductory Technical White Paper," in *ETSI White Paper*, 2014.

[6] ETSI, "Network Functions Virtualisation (NFV); Architectural Framework." https://portal.etsi.org/nfv/nfv_white_paper.pdf, 2013.

[7] Z. Bronstein, E. Roch, J. Xia, and A. Molkho, "Uniform Handling and Abstraction of NFV Hardware Accelerators," *IEEE Network*, vol. 29, pp. 22–29, May 2015.

[8] T. Lin, B. Park, H. Bannazadeh, and A. Leon-Garcia, *SAVI Testbed Architecture and Federation*, pp. 3–10. Springer International Publishing, 2015.

[9] "Elastic Cloud Compute (EC2) - AWS." [Online] Available: https://aws.amazon.com/ec2/ [Accessed: April 2017].

[10] "Microsoft Azure - Cloud Computing Platform & Services." [Online] Available: https://azure.microsoft.com/ [Accessed: April 2017].

[11] ETSI, "Network Functions Virtualisation (NFV); Infrastructure Overview," in *ETSI GS NFV-INF 001 V1.1.1*, 2015.

[12] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pp. 109–116, May 2014.

[13] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling Flexible Network FPGA Clusters in a Heterogeneous Cloud Data Center," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 237–246, ACM, 2017.

[14] T. Lin, B. Park, H. Bannazadeh, and A. Leon-Garcia, "Enabling L2 Network Programmability in Multi-Tenant Clouds," in *2017 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2017.

[15] J. Lin, Q. Zhang, B. Park, H. Bannazadeh, and A. Leon-Garcia, "MonArch: Monitoring and Analytics in Software Defined Infrastructures," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct 2015.

[16] "Data Plane Development Kit." [Online] Available: http://dpdk.org/ [Accessed: May 2017].

[17] "EC2 Instance Types." [Online] Available: https://aws.amazon.com/ec2/instance-types/ [Accessed: May 2017].

[18] "Azure Windows VM Sizes - Compute Optimized." [Online] Available: https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/sizes-compute [Accessed: May 2017].

[19] "Vivado High-Level Synthesis." [Online] Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html [Accessed: May 2017].