OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in OpenStack

Xiongzi Ge[†], Yi Liu[§], David H.C. Du[†], Liang Zhang[‡], Hongguang Guan[‡], Jian Chen[‡], Yuping Zhao[‡] and Xinyu Hu[‡]

[†]Department of Computer Science and Engineering, University of Minnesota
[§]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
[‡]Huawei Corporation
xiongzi@cs.umn.edu, liuyi@siat.ac.cn, du@cs.umn.edu, zhangliang1@huawei.com

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management

Keywords

Network Function Virtualization; Middlebox; OpenStack; FPGA

1. INTRODUCTION

Specified appliances or *middleboxes* (MBs) have been explosively used to satisfy a various set of functions in operational modern networks, such as enhancing security (e.g. firewalls), improving performance (e.g. WAN optimized accelerators), providing QoS (e.g. Deep Packet Inspection (DPI)), and meeting the requisite others [3]. Network Function Virtualization (NFV) recently has been proposed to optimize the deployment of multiple network functions through shifting the MB processing from customized MBs to software-controlled inexpensive and commonly used hardware platforms (e.g. Intel standard x86 servers) [4]. However, for some functions (e.g. DPI and Network Deduplication (Dedup), Network Address Translation (NAT)), the commodity shared hardware substrate remain limited performance. For a standard software based Dedup MB (Intel E5645, 2.4GHZ, 6 cores, exclusive mode), we can only achieve 267Mbps throughput in each core at most. Therefore, the resources of dedicated accelerators (e.g. FPGA) are still required to bridge the gap between software-based MB and the commodity hardware.

To consolidate various hardware resources in an elastic, programmable and reconfigurable manner, we design and build a flexible and consolidated framework, OpenANFV, to support virtualized acceleration for MBs in the cloud environment. OpenANFV is seamlessly and efficiently put into Openstack to provide high performance on top of commodity hardware to cope with various virtual function requirements. OpenANFV works as an independent component to manage and virtualize the acceleration resources (e.g. *cinder* manages block storage resources and *nova* manages computing resources). Specially, OpenANFV mainly has the following three features.

- **Automated Management.** Provisioning for multiple VNFs is automated to meet the dynamic requirements of NFV environment. Such automation alleviates the time pressure of the complicated provisioning and configuration as well as reduces the probability of manually induced configuration errors.
- **Elasticity.** VNFs are created, migrated, and destroyed on demand in real time. The reconfigurable hardware resources in pool can rapidly and flexibly offload the corresponding services to the accelerator platform in the dynamic NFV environment.
- **Coordinating with OpenStack.** The design and implementation of the OpenANFV APIs coordinate with the mechanisms in OpenStack to support required virtualized MBs for multiple tenancies.

2. ARCHITECTURE AND IMPLEMENTATION

Figure 1 briefly shows the architecture of OpenANFV. From the top-to-down prospective view, when a specific MB is needed, its required resources are orchestrated by OpenStack. There are sufficient northbound APIs in this open platform. Each VNF of MB is instantiated by a Virtual Machine (VM) node, running on the common x86 platform. Once deployed, these MBs aim to provide services as well as the original appliances. The role of *VNF controller* is to leverage the resource demand of VMs associated with underlying virtual resource pools.

NFV Infrastructure (NFVI) comes from the concept of *IaaS* to virtualize corresponding hardware resources. In NFVI, the *Network Functions Acceleration Platform* (NFAP) provides a heterogeneous PCIe based FPGA card which supports isolated *Partial Reconfigure* (PR) [5]. PR can virtualize and reconfigure the acceleration functions shortly in the same FPGA without affecting the exiting virtualized accelerators. Using PR, different accelerators are placed and routed in special regions.

The FPGA is divided into static and partially reconfigurable regions (PR). When the network requirements are changed, the function of PR could be replaced with a new

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631426.

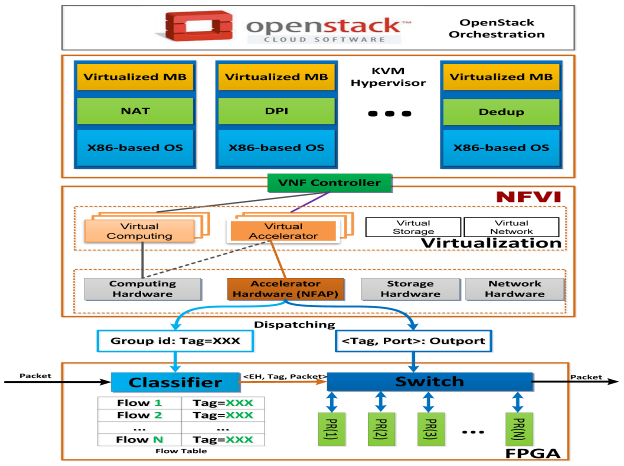

Figure 1: Brief OpenANFV architecture.

one. The PR is implemented to support different accelerators, and controlled by one VM exclusively. When one accelerator is reloading without affecting the other accelerators are not affected. The PR and VM can communicate through PCIe SR-IOV efficiently. The static region contains the shared resources (e.g. the storage and network interfaces).

The responsibility of NFAP is dispatching the rules to the two modules of FPGA, the *classifier* and the *switch*. The *classifier* identifies the flow where the rule format conforms to $\langle$*group id, tag*$\rangle$, when the flow matches the condition *group id*, classifier will add an item to the flow table in the classifier. Based on the flow table, the classifier could divide the traffic into different NFs. The switch get the MB chain configuration [$\langle Tag,Port\rangle:Outport$], and forward the encapsulated head (EH) based packet combining with the tag and income port. The packets are redirected to the MB in the chain in sequence until to the last one. The encapsulated packet with its tag can transfer more than one PRs, moreover, the tag could also support the load balancing between the NFs. Compared with FlowTags [2], we use the tag in the encapsulation to expand flexibly without affecting the field of the original packet header.

We have implemented the prototype of integrating NFAP into OpenStack following the methodology which is proposed by the ETSI standards group [1]. The K-V pair of <*vAccelerator*, *number of running vAccelerators*> is ued to track the current status of NFAP. *vAccelerator* is the aliased key to identify the PCIe device and *number of running vAccelerators* is to identify the current virtual accelerators. The scheduler has been finished in the *VNF controller* which is followed by the standard *nova scheduler*. Finally, the extended *VM.xml generation* includes allocation of a PCIe device virtual function to a VM.

3. EVALUATION AND CONCLUSION

We evaluate the prototype of NFAP in a demo FPGA cards (Altera FPGA Stratix A7, 8GB DDR3, 8M QDR, 4 SPF+). The experimental environment consists of one x86-based server (2.3GHz 8Core Intel Xeon E5, 96GB RAM, 40G NIC) with Altera FPGA card, two x86-based servers running the controller and OpenStack, respectively. The IXIA XM2(with NP8 board) is used as the source and the sink of packets. The server with the NFAP runs KVM hypervisor, and each NFAP could provide three empty PRs for the server. The VMs used for the test have the same configuration (1 vCPU, 16G RAM and 1TB Disk).

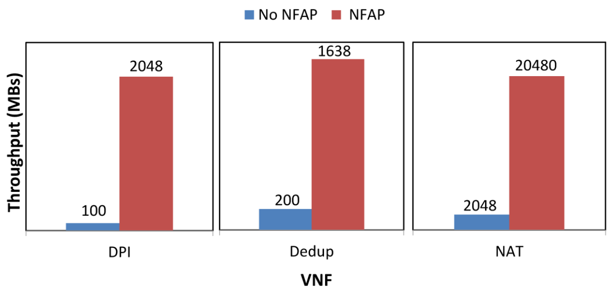

Figure 2: Performance results on throughput with or without adopting NFAP.

Our tests include three VNFs, NAT, DPI, and Dedup. Each VNF has two versions, with and without adopting NFAP. Without NFAP, the computing resources of VNFs are completely provided by *nova* in OpenStack. For NFAP assisted NAT, the software in the VM has an Openflow-like API with the NFV controller and configures the policy using a hardware abstract API. The hardware part in the NFAP offloads the flow table, the header replacement, and the checksum calculation from the VM. The packet will be merely processed in the NFAP, if the flow is matched in the flow table. For NFAP assisted DPI, we offload all the string match (Mutihash Algorithm and Bloomfilter), regular match, and the rules table in the NFAP, and the VM keeps the rule compiler and the statistics which are needed by the controller. The rule compiler compiles perl compatible regular expression to Deterministic Finite Automata (DFA) rule. For NFAP assisted Dedup, we offload the rabin hash, Marker select algorithm and chunk hash (Murmur hash). As the TCP is too complex to implement in the FPGA, the tcp-stack is still in the VM and the packets are received and sent out via the software tcp-stack. We still do some optimizations like using the Data Plane Development Kit (DPDK) driver and use space tcp-stack. As shown in Figure 2, the performance of DPI, Dedup, and NAT with adopting NFAP in OpenANFV outperforms the scheme without NFAP by 20X, 8.2X, and 10X, respectively.

4. REFERENCES

[1] Enhanced platform awareness for pcie devices. `https://wiki.openstack.org/wiki/Enhanced-platform-awareness-pcie`.

[2] Fayazbakhsh, S. K., Chiang, L., Sekar, V., Yu, M., and Mogul, J. C. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proc. 11th USENIX NSDI* (2014), USENIX Association, pp. 543–546.

[3] Gember, A., Grandl, R., Khalid, J., and Akella, A. Design and implementation of a framework for software-defined middlebox networking. In *Proceedings of the ACM SIGCOMM 2013* (2013), ACM, pp. 467–468.

[4] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., and Huici, F. Clickos and the art of network function virtualization. In *Proceedings of the 11th NSDI* (2014), USENIX, pp. 459–473.

[5] Yin, D., Unnikrishnan, D., Liao, Y., Gao, L., and Tessier, R. Customizing virtual networks with partial fpga reconfiguration. *ACM SIGCOMM Computer Communication Review 41*, 1 (2011), 125–132.