A Heterogeneous Parallel Packet Processing Architecture for NFV Acceleration

Jinshu Su*†, Biao Han*, Gaofeng Lv*, Tao Li*, Zhigang Sun*
*College of Computer, National University of Defense Technology, ChangSha, 410073, China
†National Key Laboratory of Parallel and Distributed Processing, ChangSha, 410073, China
{sjs, nudtbill, lvever, taoli, sunzhigang}@nudt.edu.cn

***Abstract*—Network function virtualization (NFV) offers a new way to design, deploy and manage networking services. It is of vital importance to exploit heterogeneous parallelism between hardware and software, in order to improve virtulization performance and quality of virtualized network services. In this poster, we propose a novel heterogeneous parallel architecture that highly exploits the parallelism inside packet processing, and implementation efficacy with hardware processing engines and software threads. We present two packet processing pipelines with three implemented VNF instances to better demonstrate the efficiency of heterogeneous parallelism in accelerating NFV. We show the performance of our proposed architecture with various virtualized requirements and traffics in a well-deployed network environment. Experimental results reveal that it can achieve accelerated NFV performance, as well as provide a wide class of VNFs to improve the quality of virtualized network services.**

***Index Terms*—network function virtualization, heterogeneous parallelism, VNF**

I. Introduction

Network function virtualization (NFV) is a network architecture concept that utilizes the virtualization technology to virtualize entire classes of network node functions into building blocks that may connect, or chain together, to create communication services. A virtualized network function, or VNF, may consist of one or more virtual machines running different software and processes, on top of standard high-volume switching and routing devices, or even cloud computing infrastructure, instead of requiring custom hardware appliances for each network function [1].

While NFV has been studied in the wider space of computer networks, exploiting parallelism to support fully virtualized network services is attracting extensive attentions of late. Until very recently, most of the proposals achieve network function parallelism by running multiple independent stacks in parallel or using co-processors in software [2] [3]. Nevertheless, current software parallelization approaches may introduce inestimable processing delay, especially when the virtualized network functions are employed by massive subscribers. In addition, hardware solutions for NFV acceleration are very challenging to achieve packet processing flexibility. Therefore, it is of vital importance to exploit heterogeneous parallelism in packet processing to improve the virtulization performance and quality of virtualized network services.

In order to achieve the above-mentioned objectives, in this poster, we propose a novel heterogeneous parallel packet processing architecture that highly exploits the parallelism inside packet processing, which is effectively implemented with hardware processing engines and software threads connected via fast I/O channels. We design and implement two packet processing pipelines with three implemented VNF instances to better evaluate the efficiency of heterogeneous parallelism in accelerating NFV. Specifically, we exploit the packet processing parallelization potentiality of Openflow and implement an extended Openflow VNF, with fine-grained packet processing flexibility. A traffic manager (TM) VNF is implemented to improve the quality of virtualized services for NFV acceleration in terms of queue dispatching, scheduling and reserved bandwidth allocation.

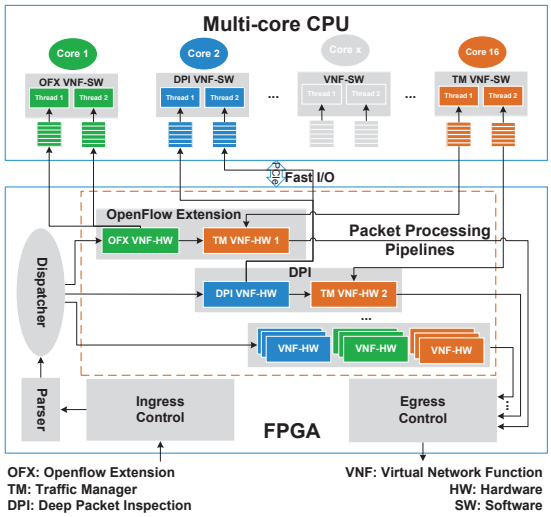


Fig. 1. The implemented NFV acceleration platform architecture

II. System Architecture

Fig.1 depicts the overview of our implemented NFV acceleration platform architecture. The core components in the platform are a FPGA-based network hardware combined with auxiliary multi-core processors.

First, in order to fully utilize the high-speed parallel processing capability of FPGA to accelerate NFV, multiple parallel packet processing pipelines are implemented in our platform. Heterogeneous parallelism can be exploited by splitting VNFs into stateful protocol processing threads in software (VNF-SW) and fast packet forwarding engines in hardware (VNF-HW). A VNF instance consists of VNF-SW threads run in individual CPU core and VNF-HW modules executed in a light FPGA shell, in which states are retrieved and configured via fast

Corresponding author is Biao Han(nudtbill@nudt.edu.cn).

978-1-7281-2700-2/19/$31.00 2019 © IEEE

I/O channels. According to certain virtualization requirements, packet processing functionalities along a specific pipeline can be dynamically orchestrated into multiple VNFs in software. In this poster, we will present two packet processing pipelines with three implemented VNFs to better demonstrate the efficiency of heterogeneous parallelism in accelerating NFV.

Second, in order to improve the quality of virtualized services for NFV acceleration in terms of queue dispatching, scheduling and reserved bandwidth allocation, each flow queue is attached with an individual per-port processing priority in TM VNF-HW, which enables dynamic allocation of queue resources by TM VNF-SW. Flow queues are organized as single-stage multiple ones. The key functionality is a preemptive priority based Deficit Round Robin (DRR) scheduling mechanism, namely PP-DRR, which is performed in a parallel and unified manner. A TM VNF instance consists of a TM VNF-SW thread in an individual CPU core and a TM VNF-HW module within a packet processing pipeline. It is feasible to deploy multiple instances in parallel to further exploit the unutilized bandwidth for differentiated qualities of services and improve the link utilization.

Third, as current version of Openflow specification does not support heterogeneous parallelism and NFV acceleration, it is beyond its capability to provide fine-grained and accelerated packet processing for NFV. In this poster, we exploit the parallel packet processing potentiality of Openflow and implement an extended Openflow VNF, named OFX VNF, which consists of an OFX VNF-HW module and multiple OFX VNF-SW threads in multi-core. Besides of ordinary capabilities, OFX VNF-SW threads can be associated to individual flows. Functionalities such as packet abstract extraction, truncation and compression can be parallelly performed in software threads, upon the Openflow based packet forwarding engines implemented in hardware. Besides, we also provide a DPI VNF in our platform, in which matching engine functionalities can be parallelly executed in both DPI VNF-HW and DPI VNF-SW [4].

III. Evaluations

The evaluation environment is set up as shown in Fig.2. In this poster, we will show the performance of our NFV acceleration platform with OFX VNF and TM VNF instances. The main equipment is our implemented NFV acceleration platform, which can support high capacity packet processing, routing, switching, and deep packet inspection, with up to 4*10G LAN/WAN/POS in standard SFP+ interfaces or 32*1GE in standard SFP interfaces. At the core of the platform are two Altera Stratix V GX FPGAs, adjacent to a main control card with two multi-core processors in it. There are 16 cores within each processor. The main evaluation tool is a SPIRENT Adtech AX/4000 Broadband Test System, which provides 1GE ports to analyze traffics go through the NFV acceleration platform via a standard SFP interface. Connected with the test system with two 1GE ports, our NFV acceleration platform receives packets via these two ports and send back to the test system via one port at rate of 1Gbps, in which network congestion will occur and our platform plans to schedule the traffic. Both of our platform and testing system are connected to a control terminal in LAN, by which configurations are managed via Command Line Interface (CLI) and testing software.

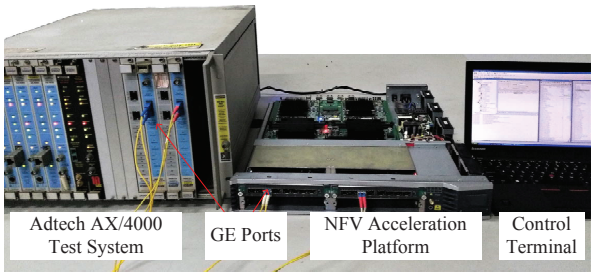
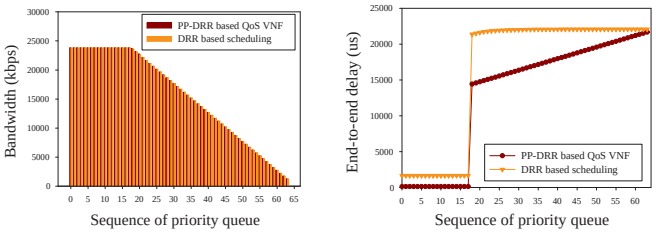


Fig. 2. The evaluation environment



(a) Bandwidth (b) End-to-end delay

Fig. 3. Performance over 64 priority queues

We use the test system to generate 64 flows at each 1GE port towards our NFV acceleration platform. It is noted that each flow is processed in a flow queue with an individual transmission priority in the generated traffic. After receiving the packets, our platform will schedule them via the proposed PP-DRR based TM VNF, and send them at an output rate of 1Gbps back to the test system. We can observe the transmission bandwidth and end-to-end delay on the control terminal. Fig.3 shows the bandwidth and corresponding end-to-end delay of PP-DRR based TM VNF and DRR based scheduling mechanism over 64 individual priority queues with descending quotas. It can be observed that our platform reduces the end-to-end delay while achieving almost the same bandwidth over all priority queues. The reason is that we always schedule the traffic flow with a higher priority through parallel packet processing between and within TM VNF-HW and TM VNF-SW. In order to evaluate the heterogeneous parallelism efficiency of the OFX VNF instances, we conduct experiments of demonstrating the procedure of deploying fine-grained flow rules to Openflow pipeline, and dynamic binding between flows and specific cores. Evaluation results can be observed by presenting the utilization rate on each core.

References

[1] Network Functions Virtualisation, http://portal.etsi.org/portal/server.pt/community/NFV/367, ISG web portal, accessed in 2019.

[2] Zhang, Yang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in Proceedings of the Symposium on SDN Research (SOSR), pp. 143-149. ACM, 2017.

[3] Sun, Chen, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu, "Nfp: Enabling network function parallelism in nfv," in Proceedings of SIGCOMM, pp. 43-56. ACM, 2017.

[4] Su, Jinshu, Shuhui Chen, Biao Han, Chengcheng Xu, and Xin Wang, "A 60GBps DPI prototype based on memory-centric FPGA," in Proceedings of the 2016 ACM SIGCOMM Conference, pp. 627-628. ACM, 2016.