Dynamic hardware acceleration of VNFs in NFV environments*

Gourav Prateek Sharma, Wouter Tavernier,
Didier Colle, Mario Pickavet
Ghent University - IMEC,
IDLab, Department of Information Technology
Ghent, Belgium
gouravprateek.sharma@ugent.be

***Abstract*—In this paper, we describe a scheme for dynamically provisioning accelerator resources to virtual network functions (VNF) in an NFV environment. The scheme involves collaboration between various NFV components like service-specific manager (SSM) and element-management-systems (EMSs) for the management of accelerator resources. Accelerator resources are dynamically allocated to VNFs based on their resource usage information. We present the performance comparison of non-accelerated and accelerated SSH-client VNFs. We also demonstrate switching of accelerator resources between the concurrently running SSH-tunnels which is triggered by a change in the nature of the data traffic flowing through SSH-tunnels.**

***Index Terms*—NFV, FPGA, accelerator, hardware, SSH, tunneling, SSM**

I. Introduction

Network services are conventionally deployed using specialized and proprietary hardware appliances called middleboxes. The objective of Network function virtualization (NFV) is to decouple the packet-processing functionality of these middleboxes from the underlying hardware so that standard IT virtualization technologies can be utilized to execute network functions on general-purpose x86 or ARM servers. NFV has enabled faster deployment of new network services along with a reduction in capital and operational expenditures. In spite of all the benefits that NFV offers, it still faces obstacles towards its widespread acceptance by telecom operators. The biggest challenge is to achieve the same virtual network function (VNF) performance as offered by its hardware counterpart [1]. To overcome this challenge, the use of hardware accelerators (GPUS, FPGAs, smartNICs) in conjunction with general-purpose processors has been advocated.

In the process of migration towards virtual packet-processing implementations from the fixed-hardware implementation, re-configurable compute platforms like FPGAs acting as hardware accelerators for VNFs are gaining a special attention. FPGAs offer the best of both worlds, i.e., the flexibility of general purpose processors and the performance of dedicated hardware boxes. Therefore, compute-intensive portions of a network function running on the CPU could be offloaded to the re-configurable accelerators running on an FPGA. In some COTS servers, a CPU can be integrated with programmable logic on the same die or it can be attached to a FPGA board via a PCIe bus.

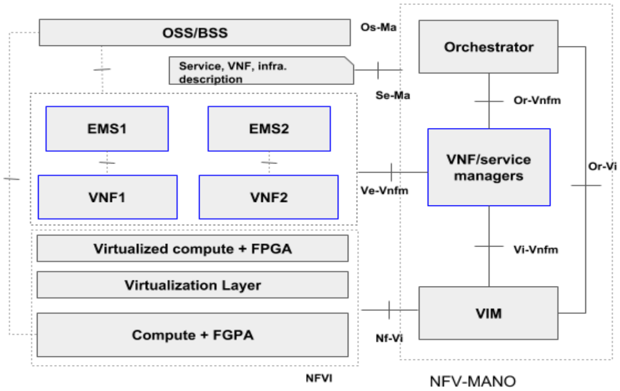

Fig. 1. ETSI's reference architecture for NFV.

The flexible and scalable hardware-acceleration of multiple VNFs in an NFV environment is still a challenge. Moreover, as hardware-accelerator resources are limited, an efficient allocation strategy is required in order to integrate them in the NFV environment. Different components of the NFV reference architecture, proposed by the ETSI, are depicted in Fig.1. The key components which are relevant to our scheme are highlighted in the blue. The VNF manager (VNFM) or service-specific manager (SSM) is responsible for life-cycle management, i.e. starting/stopping, scaling and configuring, of one or more VNFs during the service lifetime. The element management system (EMS) for each VNF and the SSM coordinate with each other to manage the service specific parameters of VNFs during their life-cycle. We have implemented an SSM which feeds on the resource utilization information of VNFs and determines which VNF to allocate available accelerator resources. We have chosen SSH-tunneling service to demonstrate a scheme for the dynamic provisioning of hardware accelerators to VNFs. Based on the real-time resource usage, we show how accelerator resources could be dynamically activated for different SSH-tunnels.

II. System Architecture and Implementation

Oftentimes, an employee of an enterprise sitting in a home needs a secure means to access network services present in a private network of its office or data-center. SSH-tunneling is the most straightforward method for creating a secure channel between a home user and a server present in the private network. An SSH-client running on the user's machine creates an encrypted tunnel to the SSH-server of the enterprise passing through the internet. After the creation of the SSH-tunnel, the SSH-client forwards packets, it receives at the listening port, to a specific mapped-port on the destination host (local port forwarding) via the SSH-server of the private network.

We have implemented and evaluated our scheme for VNFs based on SSH-client. However, system components and processes remains same for other VNFs as well. Next, we discuss the mechanism for dynamically provisioning accelerator resources by the means of SSM and other components in ETSI's NFV architecture.

A. *Service Specific Manager and NFV processes*

Fig. 2 illustrates the components and processes required for the deployment and management of hardware-accelerated VNFs in alignment with the ETSI's NFV architecture. Firstly, NFV orchestrator (NFVO) delegates the task of reserving resources for VNFs allocation to the virtual infrastructure manager (VIM). NFVO also specifies the need for any hardware accelerator cores, e.g. AES and SHA for SSH-client VNFs. A separate instance (VNF) is instantiated in order to monitor the resource utilization information, e.g. %CPU, network traffic-rate, of the service VNFs.

During the operational phase, SSM fetches the resource utilization of each VNF using the monitoring system and feeds it to the allocation logic. The allocation logic then selects the most suitable VNF to which the accelerator should be allocated. In a complete NFV system, SSM is a sub-component of NFV MANO system. In our case, SSM's role is to dynamically provision accelerator resources to VNFs based on the allocation logic. The algorithm for VNF selection is discussed in the next subsection.

After determining the next VNF to accelerate, the currently accelerated VNF is triggered to release the accelerator resource. This is followed by granting the accelerator to the selected VNF by triggering its EMS as shown in Fig 2. The required configuration of a VNF for accelerator de/allocation can be done using the interface between the EMS and the SSM.

B. *Accelerator allocation algorithm in SSM*

The heart of SSM logic lies in the algorithm for the selection of the next VNF for accelerator allocation. The allocation algorithm in our scheme is based on the hysteresis loop applied to all the VNFs of a service. The algorithm works in two steps as described using the pseudo-code in Alg. 1. The first part (lines 2-11) of the algorithm looks for a most suitable candidate VNF to allocate the accelerator for the next time slice (T). First, the algorithm loops over all running VNFs ($vnfs$), one by one, in order to select the suitable VNF for accelerator allocation. The most suitable VNF (vnf_{sel}) must have its resource usage ($vnf.usage$) higher than the pre-set upper threshold level (Th_{up}) and lowest amount of time ($vnf.ctime$) it has been allocated the accelerator in the past. The resource usage can be %CPU usage of the VNF or rate of traffic flowing through the network interface of the VNF.

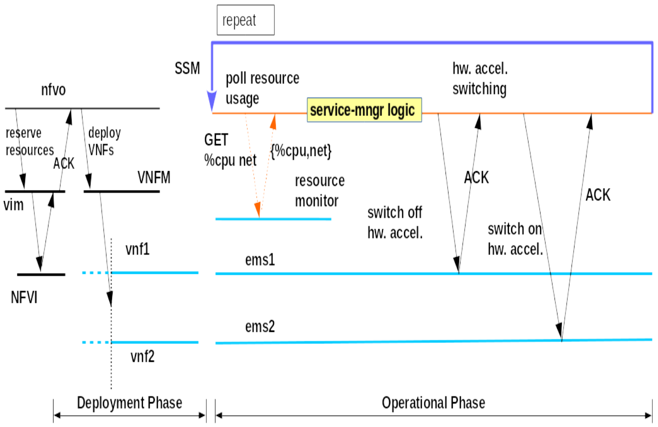

Fig. 2. Components and NFV processes involved in the accelerator allocation scheme.

In step 2 (lines 12-20), it is checked whether the allocated time for the currently accelerated VNF has expired or not. The choice of allocation time (T) determines the responsiveness and stability (no oscillations) of the algorithm and the overall overhead ($T_{sw}/(T_{sw}+T)$), where T_{sw} is the finite amount of time required for performing accelerator switching. Upon the expiry of the allocated time (T), the accelerator is taken from the currently accelerated VNF (vnf_{acc}) and granted to the VNF which was selected (vnf_{sel}) in the previous step. The time counter ($timer$) for the selected VNF is reset to the zero. The time counter and cumulative accelerator time of the selected VNF are incremented with the each pass of the loop. The algorithm tries to fairly grant all VNFs with the accelerator resource. The fairness of algorithm is a result of the fact that the VNF selection logic takes into the account cumulative accelerator allocation time ($vnf.ctime$) of all VNFs. In summary, among all the VNFs waiting for an accelerator, a VNF with the lowest $vnf.ctime$ is selected. Therefore, all VNFs are given a fair access to accelerator resources over a long period of time.

Upper threshold (Th_{up}) and lower threshold (Th_{low}) values for the CPU usage are $0.75(CPU_{max})$, and $0.65(CPU_{max})$, where CPU_{max} is the maximum CPU utilization of a VNF. The minimum accelerator allocation time (T) in our setup is 11s. This choice of threshold levels and allocation time results in a responsive and stable operation of accelerator allocation scheme.

C. *AES and SHA Acceleration in Dropbear*

For the SSH-client we have chosen Dropbear[1]. The original Dropbear implementation utilizes ciphers and hashes functions provided by libtomcrypt (cryptographic library)[2], to perform en/decryption and hashing on data packets. These functions involve multiple rounds of bit- and byte- level manipulations, e.g. XOR, substitutions, rotations, of data. A software cipher or hash function would require a CPU to execute these operations

[1] https://github.com/mkj/dropbear

[2] https://github.com/libtom/libtomcrypt

Algorithm 1 Pseudo-code for the accelerator allocation algorithm.

```
while (1) do
    for $vnf\ in\ vnfs$ do
        $instructions$
        if $(vnf.usage > Th_{up})$ and $lowestTime(vnf.time) == 1)$ then
            $vnf_{sel} \leftarrow vnf$
        else
            if $(vnf.usage < Th_{low})$ and $(vnf_{acc} == vnf)$ then
                $deallocate(vnf)$
                $timer \leftarrow 0$
            end if
        end if
    end for
    if $timer > T$ then
        $dealloc(vnf_{acc})$
        $alloc(vnf_{sel})$
        $vnf_{acc} \leftarrow vnf_{sel}$
        $timer \leftarrow 0$
    else
        $vnf_{acc}.time \leftarrow vnf_{acc}.time + 1$
        $timer \leftarrow timer + 1$
    end if
end while
```

sequentially on the input-data. On the other hand, a hardware implementation could perform these functions much faster owing to the massive parallelism available on an FPGA or ASIC.

In order to accelerate these cryptographic operations in Dropbear, FPGA based accelerators cores for AES-128/256 and SHA-256 are utilized. These cores are based on an open-source Verilog implementation [3][4]. The hardware architecture for accelerating en/decryption and hash using external hardware cores is shown in Fig. 3. For en/decryption, the hardware core is first initialized by writing keys and initialization vectors into its memory-mapped registers of the AES core. Similarly, for hash initialization, the current hash state is written to SHA-256 cores registers. Initialization is followed by the transfer of input data from the RAM memory to the BRAM of the accelerator core where cipher-text or hash is calculated. The AES/SHA core engine fetches input text from the BRAM one block (BS_{AES} = 4x32-bit words, BS_{SHA} = 16x32-bit words) at a time, processes it and then writes the processed-text back to BRAM. The progress of the core is monitored continuously by checking the progress pointer of the corresponding core, which indicates length of the input text which has been processed.

When the processing of the input-text is complete, cipher-text or hash is transferred back into the main-memory (DRAM). All data transfer tasks between the main-memory and accelerator core's BRAM are managed by the direct memory access (DMA) controller (DMAC) present on the ZYNQ processing system (PS) [2]. A kernel module[5] is used to manage DMA transfers from the user-space buffers in Dropbear to the respective cores using zero-copy mechanisms. This module creates the scatter-gather list of the memory pages corresponding to the user-space buffer and pass this list to the pl330 driver which configures the DMA controller. Next, DMAC performs the data transfer between the main-memory and BRAM without the involvement of the CPU. The hardware design for the AES-128 and SHA-256 accelerators was developed and implemented in the Vivado environment [3].

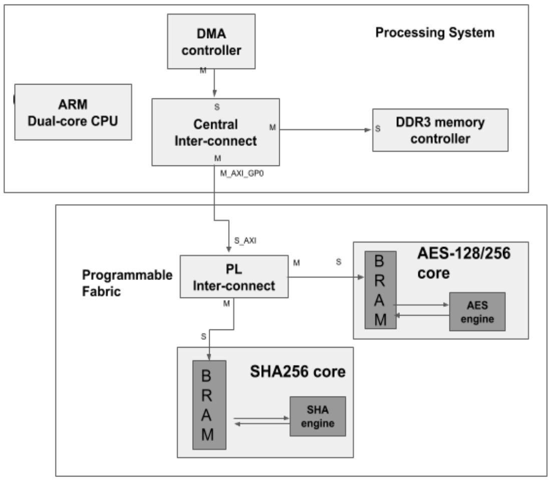

Fig. 3. Hardware design for AES en/decryption and SHA hash acceleration on PYNQ.

In order to let Dropbear offload functions to accelerator cores several modifications were added to it. The initialization and the configuration of accelerator cores requires mapping of core's address space into Dropbear's address space which is done via `mmap` system call. The transfer of input-text and the processed-text is done by performing `read` and `write` system calls to the char device exposed by the kernel module mentioned earlier. A signal handler to catch `SIGSUR1` is also included in the SSH-client. Upon the arrival of `SIGSUR1` from the kernel, Dropbear switches between the two modes, i.e., non-accelerated (software only) and hardware-accelerated.

D. Complete System and Implementation

The complete system was implemented using a laptop and a PYNQ board. PYNQ board has dual-core ARM-A9 processor (PS) and a programmable fabric (PL) on the same ZYNQ chip[6]. The PS part of the ZYNQ is used to run Ubuntu-16.04 OS. PYNQ board is attached to a laptop running Ubuntu 17.04 via an Ethernet cable. To keep our implementation simple, the orchestrator (python script) is just responsible for deploying VNFs using docker-tools (docker-client and docker-

[3] https://github.com/secworks/aes

[4] https://github.com/secworks/sha256

[5] https://github.com/jeremytrimble/ezdma

[6] http://www.pynq.io/board.html

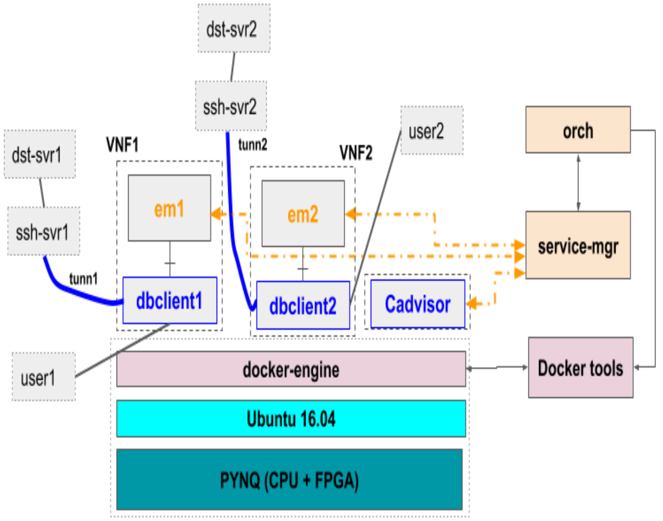

Fig. 4. System implementation for allocation of AES and SHA accelerator on PYNQ board.

daemon) and initiating the SSM. Network-functions used for establishing SSH-tunnels are the modified Dropbears SSH-clients. The complete implementation is shown in Fig. 4.

Docker-based VNFs are deployed on the PYNQ board by requesting the docker-engine running on Ubuntu-16.04 of the PYNQ board. A docker-client running on the laptop requests the docker-daemon for VNF instantiation. Each VNF for SSH-tunneling is a docker container which runs two applications:

1) Dropbear SSH-client (`dbclient`).
2) REST-ful server acting as an element manager (`em`) for the corresponding VNF.

`dbclient` is responsible for creating an encrypted channel between PYNQ board and laptop, and setting up the required port forwarding between a user on the PYNQ board to the dst-server on the laptop. The user and dst-server applications are simulated by `iperf` client and server, respectively.

Upon receiving a trigger from SSM, em sends `SIGUSR1` signal to `dbclient` process using the `kill` systen call. The signal handler in `dbclient` catches `SIGUSR1` signal and switches between the accelerated and non-accelerated modes of Dropbear.

SSM logic requires the resource usage information for all VNFs. To this end, we deploy a container-based monitoring system called Cadvisor on PYNQ. Cadvisor collects, aggregates and then exports the resource monitoring information of all the running containers to a specific port. SSM scrapes this information from Cadvisor periodically and feeds it to SSM allocation algorithm. Upon processing this information, SSM sends `GET` requests to the em of appropriate VNFs in order to grant or release accelerator resources.

III. Related works

Use of re-configurable hardware accelerator has been explored for a long time for several packet-processing applications [4] [5]. In an NFV environment, it is essential to accelerate the performance of specific VNFs running by the means of external hardware accelerators. In [6], a framework has been proposed to utilize FPGAs in order to implement complete network functions in hardware. This framework allows to build VNFs based on FPGA only and does not allow a VNF running on a CPU to offload selected tasks to FPGA accelerators. Sharing of FPGA fabric among multiple VNFs is accomplished by partial re-configuration technology. FPGAs provide a high-level of programmability as compared to ASICs but they are still expensive and less programmable as compared to COTS servers.

Li et. al. have developed dynamic hardware library (DHL), a library to abstract FPGA-based hardware accelerators for VNFs which can be accessed using a set of APIs [7]. This work focused on easing the amount of effort to access hardware accelerator from VNFs.

Byma, et. al. have proposed a framework which aggregates partial-re-configurable regions across multiple FPGAs to offer a single FPGA resource to a cloud tenant who can program its allocated region. This framework is useful for cloud service providers who would like to offer FPGAs like just like other compute resources [8].

OpenANFV is another such framework which works with OpenStack to manage and virtualize accelerator resources for VNFs requiring high-performance packet processing requirements [9]. The work done in [10] also proposed an architecture for elastic provisioning of accelerators to VNFs. A VNF can offload selected workloads to accelerator hardware modules on-demand basis. However, this work does not discuss any implementation to integrate the architecture within an NFV environment.

As accelerator resources are limited as compared to general-purpose compute resources, there is a need to efficiently allocate them among different network functions. Therefore, the challenge to dynamically allocate accelerator resource to VNFs needs to be addressed.

IV. Evaluation and Results

We evaluate our implementation in two parts. In the first part, we compare the performance of non-accelerated and accelerated VNFs in terms of their peak throughput and %CPU usage. In the second part, we verify the allocation algorithm by testing multiple VNFs with the varying traffic patterns.

A. Comparison of original and hardware-accelerated VNF

Table I shows the peak throughput values of software-only and hardware-accelerated SSH-client VNFs. We have done this comparison using two types of ciphers– AES128 and AES256. The mac-algorithm used for hashing data-packets in our evaluations is SHA-256.

As AES256 involves more number of rounds than AES128, the peak throughput of the SSH-tunnel with AES256 cipher is less than the tunnel using AES128 cipher. Moreover, one can notice a 42.1% improvement in the throughput for hardware accelerated AES128-SHA256 and 61.9% for AES256-SHA256 over non-accelerated SSH-clients. In addition to the

TABLE I
COMPARISON OF SOFTWARE AND HARDWARE CIPHERS AND HASHES.

Algorithm	Peak throughput (Mbps)	%CPU
AES128-SHA256-sw	38.8	99
AES128-SHA256-hw	55.1	84.5
AES256-SHA256-sw	31.5	99
AES256-SHA256-hw	51	86

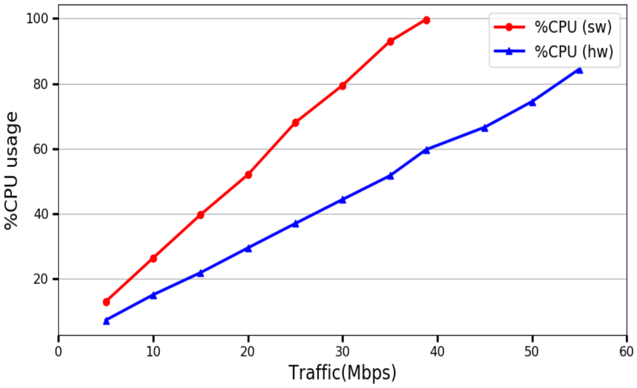

Fig. 5. Variation of %CPU usage of SSH-clients VNFS with changing traffic for non-accelerated (sw) and accelerated modes (hw).

improvement in the peak throughput, one can clearly see a reduction in %CPU usage when the VNF is accelerated. This results from the fact that CPU is relieved from performing crypto-operations which are now offloaded to accelerator cores.
The variation of %CPU utilization of the SSH-client VNF with the changing data-traffic on the tunnel is shown in Fig. 5. As the traffic on the tunnel is increased, the %CPU usage also increases. The increase in %CPU usage is because with the increase in the arrival rate the CPU spends increasingly more time in performing cipher and hash operations.

B. Dynamic accelerator allocation

The setup shown in Fig. 4 is used to verify the operation of accelerator allocation algorithm. We evaluate our implementation by instantiating `dbclient` VNFs and then loading them with time-varying traffic patterns. VNFs were pre-configured to use AES-128 cipher and SHA-256 hash while establishing SSH-sessions with the SSH-server. As a result of SSH-port forwarding, the configured ports on PYNQ board are forwarded to the laptop where the destination server is running. The home-user and destination server are simulated by `iperf` client and server applications, respectively.
We start the experiment first by deploying two VNFs on the PYNQ board and then loading them with data-traffic from two users (`iperf` clients). Fig. 6(b) shows the time variation of the peak-throughput of two SSH-tunnels. As the CPU usage of the first tunnel crosses the upper threshold (0.70) value at t=12s (Fig. 6(a)), SSM triggers em1 to let VNF1 access the accelerator resources. With the access to the accelerator, the throughput of the tunn1 is improved and its CPU usage is also reduced (Fig. 6(a)).
After a few seconds, we load the second tunnel (tunn2) with the traffic as well. At this point of time, the CPU of both tunnels is higher than the upper threshold value. However, the cumulative accelerator time for VNF2 is less than that of VNF1, because VNF1 was granted the accelerator in the previous time period. Therefore, SSM requests VNF1 to release the accelerator and which is then granted to VNF2 for a time period of about 11s, resulting in a switch-over shown at t=22s. Thereafter, the traffic remains high on both tunnels such that their CPU usages are above the threshold values, SSM grants accelerator resource to two VNFs according to their cumulative accelerator time resulting in a round-robin allocation. At accelerator switch-overs (t = 22s, 35s, 45s ..), accelerator access time for a VNF is finished and the accelerator is allocated to the other VNF.
We repeat the above experiment for two more VNFs, such that there are four concurrent SSH-tunnels established between the PYNQ board and the laptop. Each VNF corresponding to these tunnels have a CPU-share of 0.4. All the four tunnels (tunn1-tunn4) are loaded with the data-traffic and the resulting throughput and CPU usage is observed. The average CPU usage of all tunnels remains around 0.4 but a drop can be noticed during the time intervals of hardware-acceleration. In the Fig. 6(d), traffic variation for four tunnels is shown. The traffic-rate and the corresponding CPU usage of the tunn2 is highlighted in red. It can be noticed that the CPU usage of VNF2 is lowest when it has the highest throughput during the following time intervals – t=18-30, 55-67, 140-116. As the allocated accelerator time for one VNF is completed, a new VNF with lowest cumulative accelerator-time is allocated the accelerator. The accelerator allocation period (T=11s) for four tunnel and their switch-overs from one VNF to another can be observed from the Fig. 6(d).

V. CONCLUSION

Hardware accelerators are increasingly becoming a part of NFV infrastructure to accelerate packet processing perfor mance of VNFs such that SLAs are met. A mechanism is required for the flexible and dynamic provisioning of accelerator resources in an NFV scenario. SSM is the NFV-MANO component that is responsible for VNF management aspects including accelerator de-allocation and allocation tasks. We made the following observation from the evaluation of our implementation of the accelerator allocation scheme:

1) VNFs throughput improvement and a reduction in overall %CPU usage is achieved by offloading AES and SHA operations to hardware accelerator cores.
2) A dynamic provisioning of accelerator resources among multiple VNFs can be achieved based on the real-time resource usage and cumulative allocation times of VNFs.
3) The proposed accelerator allocation scheme complies with the ETSIs reference architecture for NFV.

This work can be extended by comparing the accelerator allocation algorithm based on the performance-profile of VNFs with the current reactive approach.

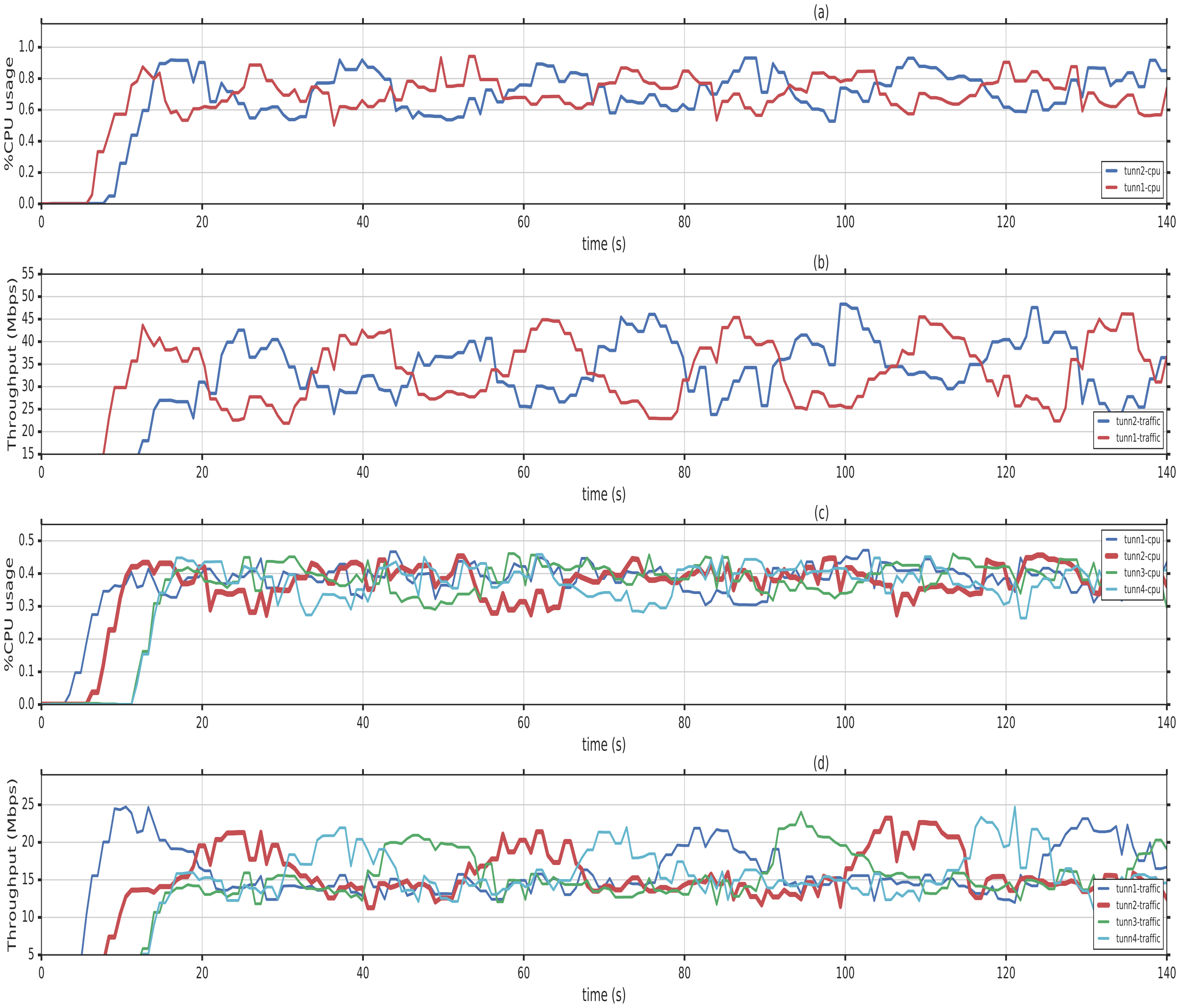

Fig. 6. Variation of (a), (c): CPU usage and (b), (d): traffic rate with time corresponding for two and four concurrent SSH-tunnels.

Acknowledgment

This work has been performed within the framework of the NGPaaS project, which is funded by the European Commission through the Horizon 2020 and 5G-PPP programs.

References

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[2] *Zynq-7000 All Programmable SoC Technical Reference Manual*, v1.10 ed., Xilinx, 2015.

[3] *Vivado Design Suite User Guide*, v2014.1 ed., Xilinx, April 2014.

[4] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters," in *11th Symposium on High Performance Interconnects, 2003. Proceedings.* IEEE, August 2003, pp. 44–51.

[5] A. Wicaksana and A. Sasongko, "Fast and reconfigurable packet classification engine in FPGA-based firewall," in *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, July 2011, pp. 1–6.

[6] C. Kachris, G. C. Sirakoulis, and D. Soudris, "Network function virtualization based on FPGAs: A framework for all-programmable network devices," *CoRR*, vol. abs/1406.0309, August 2014.

[7] X. Li, X. Wang, F. Liu, and H. Xu, "DHL: Enabling flexible software network functions with fpga acceleration," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, July 2018, pp. 1–11.

[8] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with openstack," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, May 2014, pp. 109–116.

[9] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu, "OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 353–354.

[10] L. Nobach and D. Hausheer, "Open, elastic provisioning of hardware acceleration in nfv environments," in *2015 International Conference and Workshops on Networked Systems (NetSys)*. IEEE, March 2015, pp. 1–5.