The Actual Cost of Software Switching for NFV Chaining

Marcelo Caggiani Luizelli*
Federal University of Rio Grande do Sul, Brazil
mcluizelli@inf.ufrgs.br

Danny Raz
Nokia, Bell Labs
danny.raz@nokia-bell-labs.com

Yaniv Sa'ar
Nokia, Bell Labs
yaniv.saar@nokia-bell-labs.com

Jose Yallouz*
Technion Israel Institute of Technology, Israel
jose@tx.technion.ac.il

***Abstract*— *Network Function Virtualization* (NFV) is a novel paradigm that enables flexible and scalable implementation of network services on cloud infrastructure. An important enabler for the NFV paradigm is software switching, which should satisfy rigid network requirements such as high throughput and low latency. Despite recent research activities in the field of NFV, not much attention was given to understand the costs of software switching in NFV deployments. Existing approaches for traffic steering and orchestration of virtual network functions either neglect the cost of software switching or assume that it can be provided as an input, and therefore real NFV deployments of network services are often suboptimal.**

In this work, we conduct an extensive and in-depth evaluation that examines the impact of service chaining deployments on *Open vSwitch* – the de facto standard software switch for cloud environments. We provide insights on network performance metrics such as throughput, CPU utilization and packet processing, while considering different placement strategies of a service chain. We then use these insights to provide an abstract generalized cost function that accurately captures the CPU switching cost of deployed service chains. This cost is an essential building block for any practical optimized placement management and orchestration strategy for NFV service chaining

I. Introduction

Traditional telecommunication and service networks heavily rely on proprietary hardware appliances (also called *middleboxes*) that implement *Network Functions* (NF). These appliances support a variety of NFs ranging from security (e.g., firewall, intrusion detection/prevention system) through performance (e.g., caching and proxy [1]) to wireless and voice functions (e.g., vRAN and IMS [2]).

Due to the increasing demand for network services, providers are deploying an increasing number of NFs. This requires more and more hardware-based dedicated boxes, and thus deploying new NFs is becoming a challenging and cumbersome task – which directly leads to high operational costs. The emerging paradigm of Network Function Virtualization (NFV) aims to address the aforementioned problems by reshaping the architectural design of the network [2]. Essentially, NFV leverages traditional virtualization by replacing NFs with software counterparts that are referred to as Virtual Network Function (VNF). Virtualization technologies enable to consolidate VNF onto standard commodity hardware (e.g. servers, switches, and storage), and support dynamic situations in a flexible and cost-effective way (e.g., service provisioning on demand).

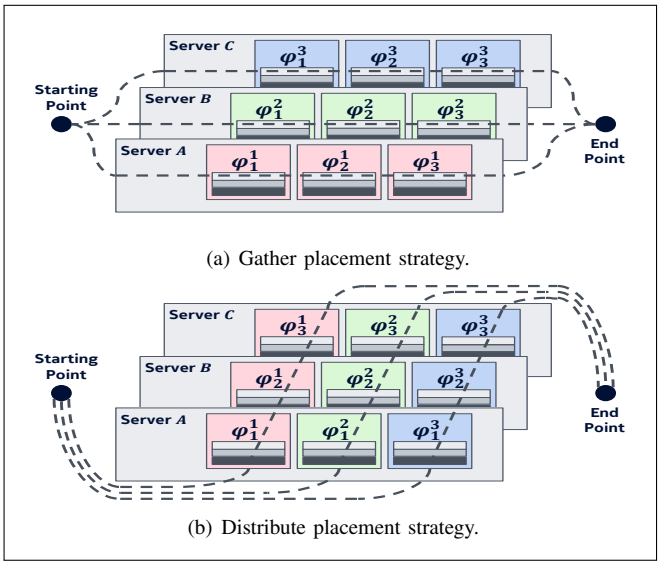
In current hardware-based networks, operators are required to manually route cables and compose physical chains of middleboxes in order to provide services, a problem that is known as *Service Function Chaining* (SFC) [3]. The manual composition of SFC is error prone and static in nature (i.e., hard to change or physically relocate), and therefore expensive. On the other hand, NFV-based service chaining (*forwarding graph*), enables flexible placement and management of services, which allows a much more efficient utilization of resources, since the same computation resources can be consumed by different NFs in a dynamic way. Thus, a key expected success criteria for NFV is efficient resource utilization and reduced cost of operation.

Placement of service chain in NFV-based infrastructure is not trivial, and introduces new challenges that need to be addressed. First, NFV is an inherently distributed network design based on small cloud nodes spread over the network. Second, even when considering a single (small) data center, network services might face performance penalties and limitations on critical network metrics, such as throughput, latency, and jitter, depending on how network functions are chained and deployed. This is one of the most interesting challenges for network providers in the shift to NFV, namely identify good placement strategies that minimize the provisioning and operation cost of deploying a service chain.

OpenStack [4] is the most popular open source cloud orchestration system, its scheduling (placement) component is called `nova-scheduler`, and can be utilized to place a sequence of VNFs. When doing so the resulting placement objective can be either load balancing (i.e., distributing VNFs between resources) or energy conserving (i.e., gathering VNFs on a selected resource). Consider the two placement strategies presented in Figure 1. In this example, we are given three identical servers A, B and C, and three identical service chains $\{\varphi^1, \varphi^2, \varphi^3\}$. Each service chain φ^i (for $i = 1, 2, 3$) is tagged with a fixed amount of required traffic and is composed of three chained VNFs: $\varphi^i = \langle \varphi^i_1 \rightarrow \varphi^i_2 \rightarrow \varphi^i_3 \rangle$. Figure 1(a) illustrates the first placement strategy (referred to as "gather"), where all VNFs composing a single chain are deployed on the same server. Note that in the gather case the majority of traffic steering is done by the server's internal virtual switching.

*Work done while in Nokia, Bell Labs.

978-3-901882-89-0 @2017 IFIP



(a) Gather placement strategy.

(b) Distribute placement strategy.

Fig. 1: Example of strategies to deploy 3 given service chains.

Figure 1(b) illustrates the second placement strategy (referred to as "distribute"), where each VNF is deployed on a different server, and therefore the majority of traffic steering is done between servers by external switches. One can immediately see that these two placement strategies require from the VNF the same amount of resources to operate, however differ in the cost of their software switching (in terms of CPU consumption). It is also not completely clear how would the network perform for each of the placement strategies, with respect to metrics such as throughput and latency.

In order to decide which of the placement strategy is better, we need to identify the specific optimization criteria of interest. In this paper we focus on accurately estimating the virtual switching cost. For NFV-based infrastructure, virtual switching is an essential building block that on one hand enables flexible communication between VNFs; but on the other hand, comes with a cost of resource utilization. Therefore assessing, understanding, and crafting a monolithic CPU cost function for software switching in a NFV-based environment is an extremely important task and a crucial step in order to: *(i)* efficiently guide VNF placement in a NFV-based infrastructure; *(ii)* understand how software switching impacts deployed services, namely analyze and comprehend how throughput-sensitive services behave; *(iii)* whenever possible, reduce the costs to the NFV provider and ultimately, foster the development of cost optimized deployment solutions.

In this paper, we develop a model to estimate the cost of software switching for different placement strategies over NFV-based infrastructure. We conduct an extensive and in-depth evaluation, measuring the performance and analyzing the impact of deploying a service chain on *Open vSwitch* – the de facto standard software switch for cloud environments [5]–[7]. Based on our evaluation, we then continue to craft a generalized abstract cost function that accurately captures the CPU cost of network switching. The main contributions of this paper can be summarized as follows: ***(i)* comprehensive evaluation of placement strategies of service chains** – based on a real NFV-based infrastructure, we examine our placement strategies, and assess performance metrics such as throughput, packet processing, and resource consumption[1]; and ***(ii)* model the cost of network switching** – given an arbitrary number of service chains, our model accurately predicts the CPU cost of the network switching they require.

The rest of the paper is organized as follows. In Section II we define our model and in Section III we provide an extensive and in-depth evaluation of software switching performance. In Section IV we analyze the results and build an abstract generalized cost function for our model. In Section V we discuss related works, and finally in Section VI we conclude our work and provide future directions.

II. Model Definition

In this section we define our model to estimate the cost of software switching for different placement strategies.

For a server S we denote by S^t the maximum throughput that server S can support (i.e., the wire limit of the installed NIC). Following the recommended best practice for virtualization intense environment ([8]), NFV servers require to be configured with two disjoint sets of CPU-cores, one for the hypervisor and the other for the VNF to operate. We denote by S^h (and S^v) the number of CPU-cores that are allocated and reserved solely for the hypervisor (and guests, respectively) to operate. The total number of CPU-cores installed in server S is denoted by S^c. Throughout this paper, unless explicitly saying otherwise, we assume that we are given a set of k servers $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$.

A *service chain* is defined to be an ordered set of VNFs $\varphi = \langle \varphi_1 \rightarrow \varphi_2 \ldots \rightarrow \varphi_n \rangle$. We denote by $\varphi|^n$ the length of the service chain, and define $\varphi|^p$ (and $\varphi|^s$) to be the number of packets per second (and average packet size, respectively) that service chain φ is required to process. Note that $\varphi|^p$ and $\varphi|^s$ are properties of service chain φ, namely given two service chains φ and ψ if $\varphi|^p \neq \psi|^p$ or $\varphi|^s \neq \psi|^s$ then $\varphi \neq \psi$. Throughout the paper, unless explicitly saying otherwise, we assume that we are given a set of m identical service chains $\Phi = \{\varphi^1, \varphi^2, \varphi^3, \ldots \varphi^m\}$, i.e. $\forall_{i,j=1..m} : \varphi^i = \varphi^j$.

We define $\mathcal{P} : \Phi \rightarrow S^k$ to be a *placement function* that for every service chain $\varphi \in \Phi$, maps every VNF $\varphi_i \in \varphi$ to a server $S_j \in \mathcal{S}$. We identify two particularly interesting placement functions:

(i) $\mathcal{P}_g$ – we call *gather* placement, where for every service chain the placement function deploys all VNFs $\varphi_i \in \varphi$ on the same server, namely:

$$\forall_{\varphi \in \Phi} \forall_{\varphi_i, \varphi_j \in \varphi} : \quad \mathcal{P}_g(\varphi_i) = \mathcal{P}_g(\varphi_j)$$

(ii) $\mathcal{P}_d$ – we call *distribute* placement, where for every service chain the placement function deploys each VNFs $\varphi_i \in \varphi$ on a different server, namely:

$$\forall_{\varphi \in \Phi} \forall_{\varphi_i, \varphi_j \in \varphi} : \quad i \neq j \rightarrow \mathcal{P}_d(\varphi_i) \neq \mathcal{P}_d(\varphi_j)$$

[1]Unlike other performance studies, our goal is *not* to delivers the best performance but rather to evaluate the performance of a given NF configuration.

As explained above, both placement functions being considered, follow OpenStack and the objective goal implemented by `nova-scheduler` which can be either load balancing that distributes VNFs between resources, or energy conserving that gathers VNFs on a selected resource. Figure 1 illustrates our deployment strategies. Figure 1(a) depicts deployment of VNFs that follows the gather placement functions $\mathcal{P}_g$, and Figure 1(b) depicts deployment of VNFs that follows the distribute placement functions $\mathcal{P}_d$.

For a given placement function $\mathcal{P}$ that deploys all service chains in Φ on the set of servers $\mathcal{S}$, we define $\mathcal{C} : \mathcal{P} \rightarrow (\mathbb{R}^+)^k$ to be a *cpu-cost functions* that maps placement $\mathcal{P}$ to the required CPU per server. We say the cpu-cost function is *feasible* if there are enough resources to implement it. Namely, cpu-cost function $\mathcal{C}$ is feasible with respect to a given placement $\mathcal{P}$ if for every server $S_j \in \mathcal{S}$, the function do not exceed the number of CPU-cores installed in the server: $\forall_{S_j \in \mathcal{S}} : \mathcal{C}(\mathcal{P})_j \leq S_j^c$. Note that our main focus is the evaluation of the cost-functions with respect to the hypervisor performance, and therefore we evaluate deployments of VNFs that are fixed to do the minimum required processing, i.e. forward the traffic from one virtual interface to another.

III. Deployment Evaluation

In this section, we evaluate the performance of software switching for NFV chaining w.r.t. metrics such as throughput, CPU utilization, and packet processing. We describe our environment setup, followed by a discussion on performance metrics and possible bottlenecks. We evaluate two types of OvS installations: Linux kernel and DPDK; and compare between the two types of VNF placements: gather placement function $\mathcal{P}_g$, versus distribute placement function $\mathcal{P}_d$.

A. Setup

Our environment setup consists of two high-end HP ProLiant DL380p Gen8 servers, each server has two Intel Xeon E5-2697v2 processors, and each processor is made of 12 physical cores at 2.7 Ghz. One server is our *Device Under Test* (DUT), and the other is our traffic generator. The servers have two NUMA nodes (Non-Uniform Memory Access), each has 192 GBytes RAM (total of 384 GBytes), and an Intel 82599ES 10 Gbit/s network device with two network interfaces (physical ports). We disabled HyperThreading in our servers in order to have more control over core utilization.

In both types of OvS installations (Linux kernel and DPDK), we isolate CPU-core and separate between two disjoint sets CPU-cores: $S^c = 24 = S^h + S^v$, i.e. CPU-cores used for management and for allocating resources for the VNFs to operate. This a priori separation between these two disjoint sets of CPU-cores plays an important role in the behavior of CPU consumption (and packet processing). In our experiments the size of the disjoint set of cores that are given to the hypervisor (S^h) varies between 2 to 12 physical cores, while the remainder is given to the VNFs (i.e. S^v ranges between 22 to 12). The exact values depend on the type of installation (Linux kernel or DPDK).

The DUT is installed with CentOS version 7.1.1503, Linux kernel version 3.10.0. All guests operating system are installed with Fedora 22, Linux kernel version 4.0.4 – running on top of qemu version 2.5.50. Each VNF is configured to have a single virtual CPU pinned to a specific physical core and 4GBytes of RAM. Network offload optimizations (e.g., `TSO`, `GSO`) in all network interfaces are disabled in order to avoid any noise in the analysis and provide a fair comparison between the two types of OvS installations. We evaluated Open vSwitch version 2.4 in both kernel mode and DPDK. For the latter, we compiled OvS against DPDK version 2.2.0 (without shared memory mechanisms – in compliance with rigid security requirements imposed by NFV). Packet generation is performed on a dedicated server that is interconnected to the DUT server with one network interface to send data, and another network interface to receive. In all experiments, we generate the traffic with Sockperf version 2.7 [9], that invokes 50 TCP flows.

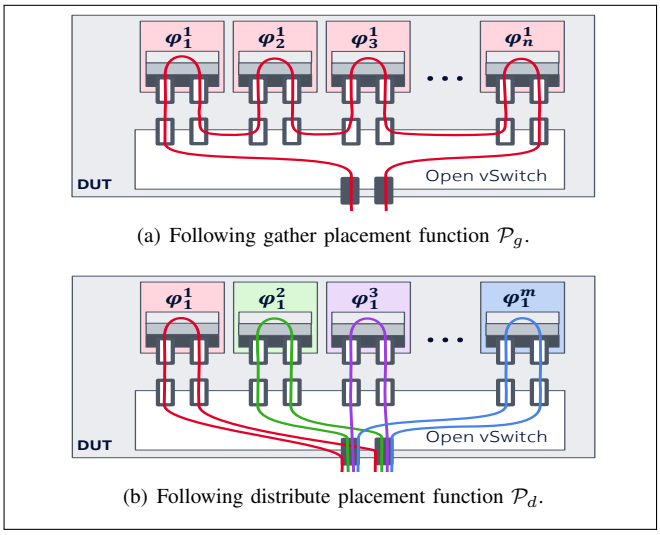
Note that our objective is to provide an analytic model that captures the cost of software switching. In order to be able to examine and provide a clear understanding of the parameters that impact the cost of software switching, we need to simplify our environment by removing optimizations such as network offloads, and fixing resource allocation. For achieving optimal performance, the reader is referred to [8].

We evaluate the placement functions defined in Section II (i.e., the gather placement function $\mathcal{P}_g$ and the distribute placement function $\mathcal{P}_d$) on our DUT as follows. We vary the number of simultaneously deployed VNFs from 1 to 30 in each of the placement functions. All VNFs forward their traffic between two virtual interfaces, each on a different subdomains, relaying on IP route forwarding.

Figure 2 illustrates deployments of VNFs on a single server (our DUT). Figure 2(a) depicts the traffic flow of the gather placement function $\mathcal{P}_g$. Ingress traffic is arriving from the physical NIC to the OvS that forwards it to the first vNIC of the first VNF. The VNF then returns the traffic to the OvS through its second vNIC, and the OvS forwards the traffic to the following VNF, composing a chain that eventually egress the traffic through the second physical NIC. On the other hand, Figure 2(b) depicts the traffic flow of the distribute placement function $\mathcal{P}_d$. For this placement, it is the responsibility of the traffic generator to spread the workload between the VNFs. Ingress traffic arriving from the physical NIC to the OvS that forwards it to one of the VNFs through its first vNIC. The VNF then returns the traffic to the OvS through its second vNIC, that egress the traffic through the second physical NIC.

B. Evaluating Packet Intense Traffic

In order to discuss the utilization of resources, we measure and analyze the results of our DUT for several configurations, while receiving intense network workload. To generate an intense network workload we generate traffic where each packet is composed of 100Bytes (avoid reaching the NIC's wire limitation). We examine the behaviour of throughput, packet processing and CPU consumption for increasing chain size, using both placement functions, i.e. $\mathcal{P}_g$ and $\mathcal{P}_d$.



(a) Following gather placement function $\mathcal{P}_g$.

(b) Following distribute placement function $\mathcal{P}_d$.

Fig. 2: Given a set of m service chains Φ, illustrating deployment of VNFs on a single server (our DUT).

The six graphs in Figure 3 (and Figure 4) depict CPU consumption, packet processing, and throughput performance on a DUT that is installed with kernel OvS (and DPDK-OvS, respectively). The three graphs on the top (Figures 3(a), 3(b), and 3(c) for kernel OvS and Figures 4(a), 4(b), and 4(c) for DPDK-OvS) present the results following the gather placement function $\mathcal{P}_g$ and the three graphs on the bottom (Figures 3(d), 3(e), and 3(f) for kernel OvS and Figures 4(d), 4(e), and 4(f) for DPDK-OvS) present the results following the distribute placement function $\mathcal{P}_d$.

We measure throughput by aiming at the total of traffic that the traffic generator successfully sends and receives after routing through the DUT. To measure packet processing we aggregate the number of received packets (including TCP acknowledgments) in all interfaces of the OvS (including both NICs and vNICs). For CPU consumption, we present the results for both the CPU-cores allocated to the hypervisor to manage and support the VNF and CPU-cores allocated for the VNFs to operate.

1) Packet Processing and Throughput: A key factor in the behavior of our environment is the a priori separation between two disjoint sets of CPU-cores (S^h and S^v). Thus, in our experiments, we vary values of CPU-cores that are allocated to the OvS (hypervisor). For ease of presentation, we show only several selected values (annotated HV in the figures).

Figure 3(a) depicts the average throughput for gather placement function $\mathcal{P}_g$, and Figure 3(d) for distribute placement function $\mathcal{P}_d$. For the case of distribute placement function $\mathcal{P}_d$, the more VNFs we deploy on the server, the more the average throughput is increased, while for the gather placement function $\mathcal{P}_g$ the more the average throughput is decreased. Figures 3(b) and 3(e) depict packet per second for both our placement function $\mathcal{P}_g$ and $\mathcal{P}_d$. The results show that OvS can seamlessly scale to support 5-10 VNF, however at that point OvS reaches saturation, and we observe mild degradation when continuing to increase the number of deployed VNFs.

Figures 4(a) and 4(d) (and Figures 4(b) and 4(e)) depict average throughput (and packet per second, respectively) of our two placement function $\mathcal{P}_g$ and $\mathcal{P}_d$, on a DUT that is installed with OvS-DPDK. The behavior of OvS-DPDK is similar to that of the kernel OvS, with the exception of having better network performance.

2) CPU Consumption: Figures 3(c) and 3(f) depict the average CPU consumption of OvS in Kernel mode. For both placement functions, the CPU consumption of the VNFs is bounded by the number of allocated cores for management S^h. A tighter bound of the CPU consumed by the VNFs (for networking) is a function of the CPU consumed by the hypervisor to manage the traffic, namely the total CPU consumed by the VNF is bounded by the total CPU consumed by the hypervisor in order to steer the traffic to the VNF.

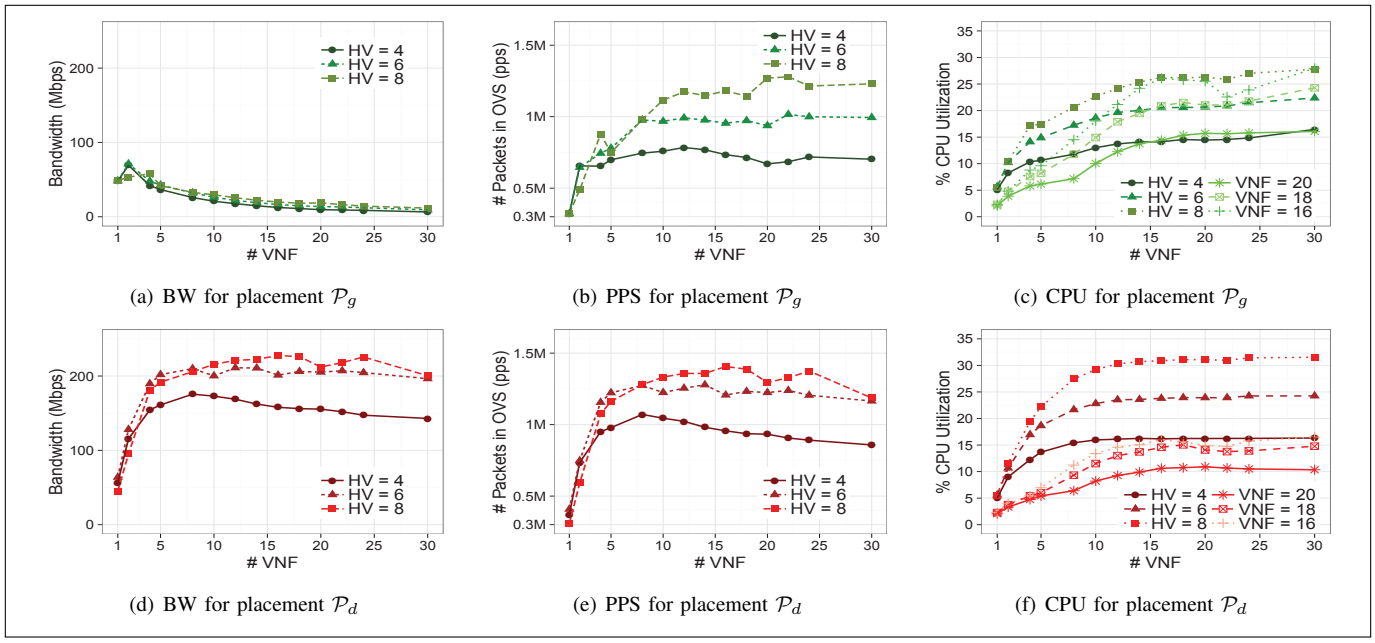
Comparing CPU utilization between the two placement functions, we observe that both placement strategies behave similarly for short service chains with little traffic. However, for longer service chains with increasing traffic requirements the behaviour of the two placements differs. Namely, in the gather placement, the CPU consumed by the VNF is almost identical to the CPU consumed by the hypervisor, whereas in the distribute placement the CPU consumed by the VNFs are 70% to 50% of the CPU consumed by the hypervisor. This observation suggests that in order to achieve CPU cost efficient placement, different traffic might require different placement strategies (as will be shown in the results of our analytical analysis in Section IV).

Figures 4(c) and 4(f) depict the average CPU consumption of DPDK-OvS. For both placement functions, the CPU consumed by the hypervisor is fixed and derived from the DPDK poll-mode driver (with the additional CPU-core for the management of other non-networking provision tasks). Comparing CPU utilization between the two placement functions shows again that both behave the same for small chains with little traffic, however for bigger chains with increasing traffic requirements the behaviour of the two placements defers.

C. Evaluating Throughput Intense Traffic

We reiterate the experiment presented in Subsection III-B in the context of achieving maximum throughput. Note that as opposed to the previous section where we wanted to examine the CPU cost of the transferred traffic, here our goal is solely to maximize our throughput. In order to discuss maximum throughput, we measure and analyze the results of our DUT for several configurations, while receiving maximum transferable unit, i.e. we generate traffic where each packet is composed of 1500Bytes. We examine the behaviour of throughput for increasing chain size, using both placement functions $\mathcal{P}_g$ and $\mathcal{P}_d$. Since the behaviour of CPU consumption, and packet processing are similar to the behaviour observed for 100Bytes packet (in bigger scale), we omit their presentation.

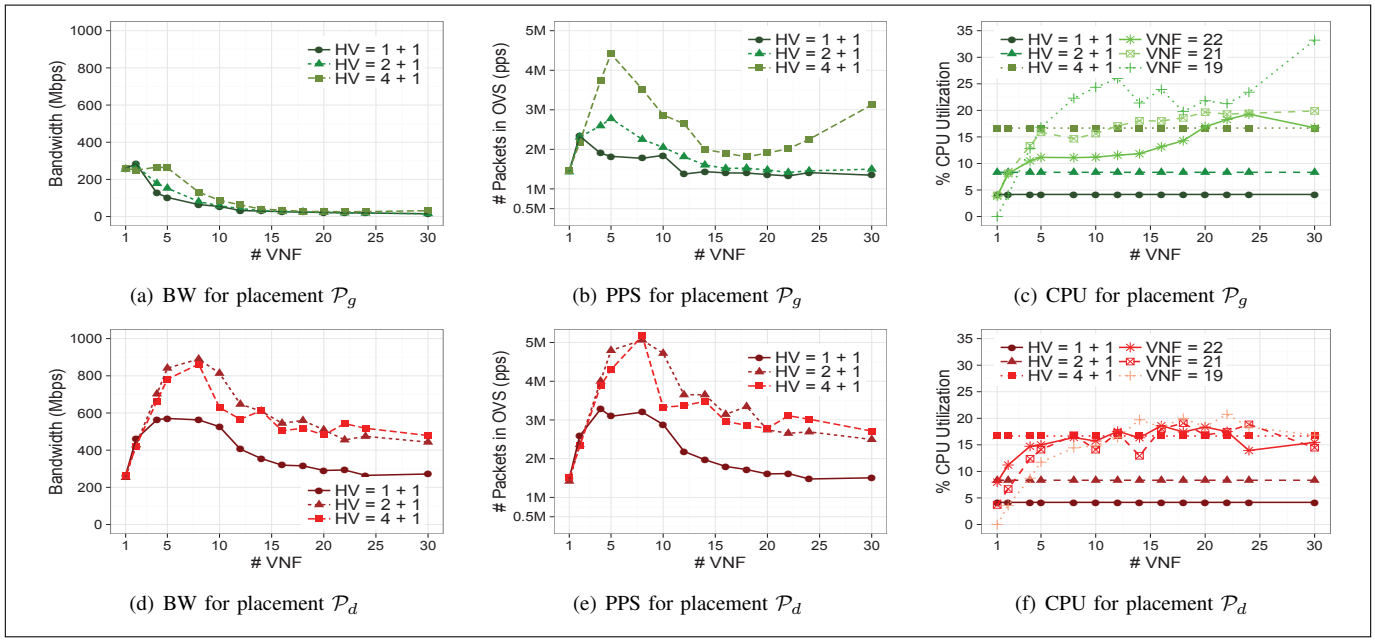
Figure 5 (and Figure 6) depicts throughput performance on a DUT that is installed with kernel OvS (DPDK-OvS, respectively). Both Figures 5(b) for kernel-OvS, and 6(b)



(a) BW for placement $\mathcal{P}_g$ (b) PPS for placement $\mathcal{P}_g$ (c) CPU for placement $\mathcal{P}_g$

(d) BW for placement $\mathcal{P}_d$ (e) PPS for placement $\mathcal{P}_d$ (f) CPU for placement $\mathcal{P}_d$

Fig. 3: Experiment results showing throughput, packet processing and CPU consumption for traffic generated in 100Bytes packets, that is examined over increasing size of chain, on a DUT that is installed with kernel OvS.

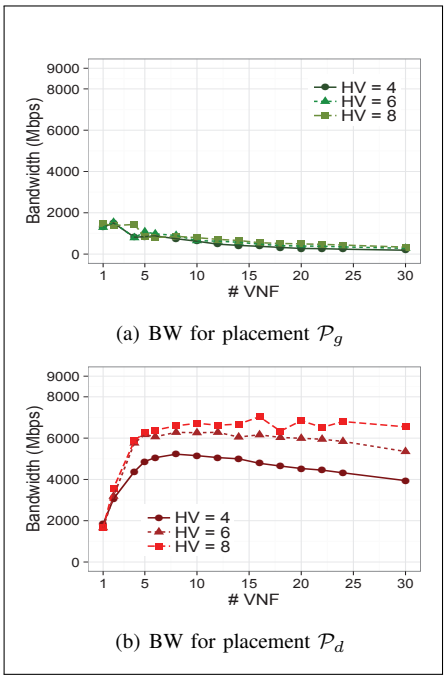


(a) BW for placement $\mathcal{P}_g$ (b) PPS for placement $\mathcal{P}_g$ (c) CPU for placement $\mathcal{P}_g$

(d) BW for placement $\mathcal{P}_d$ (e) PPS for placement $\mathcal{P}_d$ (f) CPU for placement $\mathcal{P}_d$

Fig. 4: Experiment results showing throughput, packet processing and CPU consumption for traffic generated in 100Bytes packets, that is examined over increasing size of chain, on a DUT that is installed with DPDK-OvS.

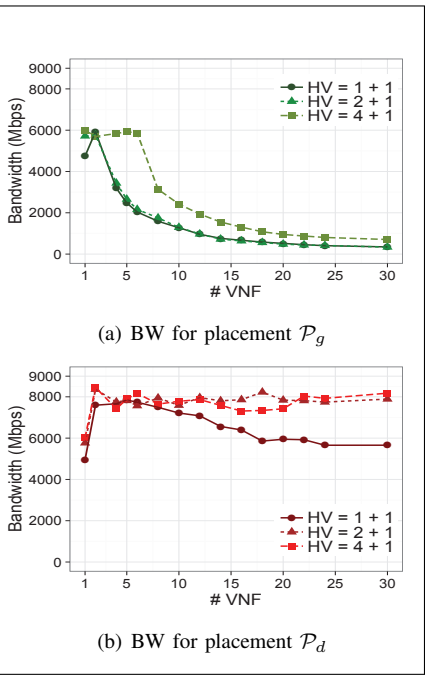
for DPDK-OvS show that the average throughput can scale up as long as the server is not over-provisioning resources (when deploying 1-5 VNFs). For the case of kernel OvS the bottleneck is the packet processing limit, while the NIC's wire limit is the bottleneck for the case of DPDK-OvS. The same effect can also be seen in Figures 5(a) and 6(a) where again as long as the server is not over-provisioning resources, the chain of VNFs is able to forward ∼0.8-1.5 Gbit/s in the case of kernel OvS, and ∼6 Gbit/s in the case of DPDK-OvS. Note that DPDK-OvS does not reach its packet processing limit (as it was seen in the case of 100-Byte packets – Subsection III-B). Instead, the observed limit is induced by the amount of packet processing that a single CPU core (allocated for the VNF) can perform (∼6 Gbit/s). This bottleneck can be



(a) BW for placement $\mathcal{P}_g$

(b) BW for placement $\mathcal{P}_d$

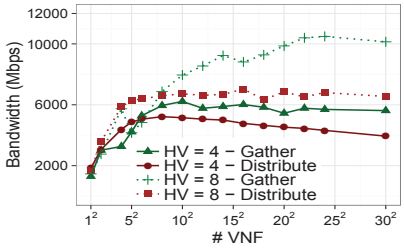
Fig. 5: Throughput for traffic generated in 1500Bytes packets, that is examined over increasing size of chain, on a DUT that is installed with kernel-OvS



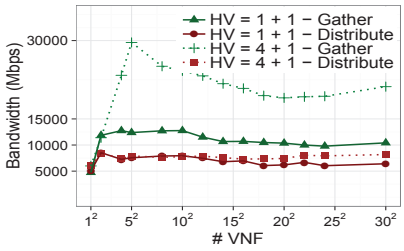
(a) BW for placement $\mathcal{P}_g$

(b) BW for placement $\mathcal{P}_d$

Fig. 6: Throughput for traffic generated in 1500Bytes packets, that is examined over increasing size of chain, on a DUT that is installed with DPDK-OvS



(a) Kernel OvS



(b) DPDK-OvS

Fig. 7: Analysis of multiple servers, showing total throughput for traffic generated in 1500Bytes packets, that is examined over increasing size of chain

mitigated if VNFs are set to have more than a single vCPU – that are configured to enable Receive Side Scaling (RSS).

So far, we presented the average throughput of a single server (our DUT). A naive straightforward analysis might lead to the wrong conclusion that the distribute placement function $\mathcal{P}_d$ outperforms the gather placement function $\mathcal{P}_g$. As can be seen in Figures 7(a) and 7(b), this is not the case. Figures 7(a) and 7(b) present the average overall throughput estimated by the model defined in Section II on a set of many servers that are installed with kernel OvS (DPDK-OvS, respectively). The gather placement function $\mathcal{P}_g$ has the advantage of being able to break the traffic between several different independent servers, whereas the distribute placement function $\mathcal{P}_d$ is bound by the wire capacity.

IV. Monolithic Cost Function

Section III focuses on exhaustive evaluations, analyzing throughput, OvS packet processing and CPU consumption for different placement functions and software switching technologies (kernel OvS and DPDK-OvS). In the following, we describe how we build the generalized abstract cpu-cost functions, and then present and discuss the results.

A. Building an Abstract Cost Function

Based on the measured results on a single server (presented in Figures 3 - 4, and 5 - 6), we are now ready to craft an abstract generalized cost function that accurately captures the CPU cost of network switching.

We iterate the following process for both kernel OvS and DPDK-OvS. For each placement function ($\mathcal{P}_g$ or $\mathcal{P}_d$) and for each packet size (100Bytes or 1500Bytes), we split the construction and build a set of sub-functions that compose the cpu-cost function, namely $\mathcal{C} = \{\mathcal{C}_{gs}, \mathcal{C}_{ds}, \mathcal{C}_{gb}, \mathcal{C}_{db}\}$ where $\mathcal{C}_{gs}$ and $\mathcal{C}_{gb}$ are sub-functions for placement $\mathcal{P}_g$ and packet size 100Bytes and respectively 1500Bytes, and $\mathcal{C}_{ds}$ and $\mathcal{C}_{db}$ are sub-functions for placement $\mathcal{P}_d$ and packet size 100Bytes and respectively 1500Bytes.

Per each sub-function and for all measured service chain length, we sample the throughput (Figures 3(a) - 3(d), 4(a) - 4(d), 5(a) - 5(b), and 6(a) - 6(b)) to estimate the amount of packets that a single server can process, and correlate between the service chain length and the amount of packets. Next, we correlate the resulting packet processing, with the measured CPU consumption (Figures 3(c) - 3(f), 4(c) - 4(f)).

Finally, after extracting a 3-dimensional correlation between *(i)* service chains length; *(ii)* packet processing; and *(iii)* CPU consumption, we use the results to extract a set of cpu-cost functions using logarithmic regression, as follows:

$$log(\mathcal{C}_X) = \alpha \cdot log(\varphi|^n) + \beta \cdot log(\varphi|^p) + \gamma$$

Where $X \in \{gs, ds, bd, db\}$, namely gather or distribute placements for 100Bytes or 1500Bytes size of packets. Table I lists per each sub-function the coefficients α and β, and the constant factor γ. Given a service chain φ, the set of sub-

	Kernel OvS			DPDK-OvS		
	α	β	γ	α	β	γ
$\mathcal{C}_{gs}$	0.586	0.858	-1.789	0.370	0.467	1.543
$\mathcal{C}_{ds}$	0.660	0.243	-2.661	0.217	0.091	3.795
$\mathcal{C}_{gb}$	0.752	0.979	-3.856	0.478	0.578	0.194
$\mathcal{C}_{db}$	1.009	0.268	-7.176	0.157	0.109	4.718

TABLE I: Coefficients α and β, and the constant factor γ, per each cpu-cost sub-function.

functions $\mathcal{C}$ estimates the total required CPU consumption on all servers.

B. Insights

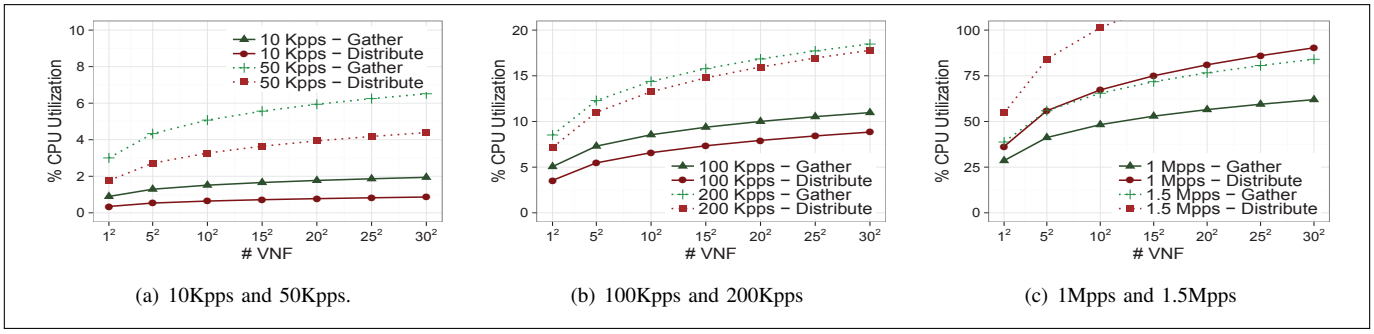
The values presented in Table I reflect the real CPU cost of the various deployments, but they provide very little insight regarding our motivation question (see Figure 1). In order to get a real understanding of this cost we provide graphs that depict the CPU cost for various service chains characterized by the length of the chain $\varphi|^n$, and the amount of packets to process $\varphi|^p$.

Figures 8(a), 8(b), and 8(c) depict the CPU cost for both placement functions when increasing the number of VNFs (and also the number of service chains) on servers that are installed with kernel-OvS, and traffic is received in large packets (1500Bytes per packet).

In all graphs, the CPU consumption is the total amount of CPU required on all physical machines to support the service chain (where 100% represents all CPU-core on all servers). For service chains with low packet processing requirements (10 Kpps - 50 Kpps), the cpu-cost function of the distribute placement $\mathcal{C}_{db}$, outperforms its gather placement counterpart $\mathcal{C}_{gb}$. However, as the requirement for packet processing increases (100 Kpps to 1.5 Mpps), the behaviour turns over and favors the gather placement cpu-cost function $\mathcal{C}_{gb}$.

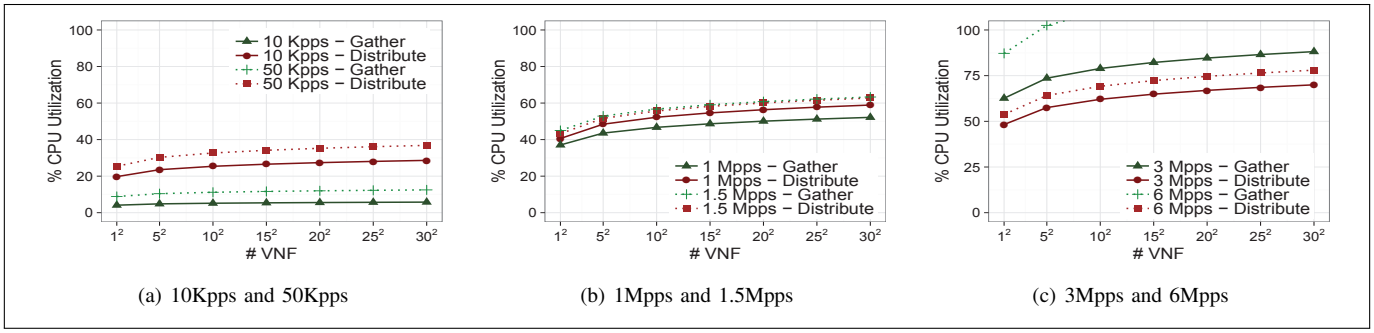
In turn, Figures 9(a), 9(c), and 9(b) depict the CPU cost for both placement functions when increasing the number of VNFs (and also the number of service chains) on servers that are installed with DPDK-OvS, and traffic is received in 1500Bytes per packet. In this case the behaviour changes. For service chains with low packet processing requirements (10Kpps - 50Kpps), the cpu-cost function of the gather placement $\mathcal{C}_{gb}$, outperform its distribute placement counterpart $\mathcal{C}_{db}$. However, as the requirement for packet processing increases (1Mpps to 6Mpps), the behaviour turns over and favors the distribute placement cpu-cost function $\mathcal{C}_{db}$. Both results presented above show that deciding which of the placement strategy is better, depends on the required demand of packets to process, where the exact dependency varies according to the technology used.

Next we examine the cpu-cost function by varying the demand (i.e. required packets to process) of a service chain, for arbitrarily selected few service chain lengths.



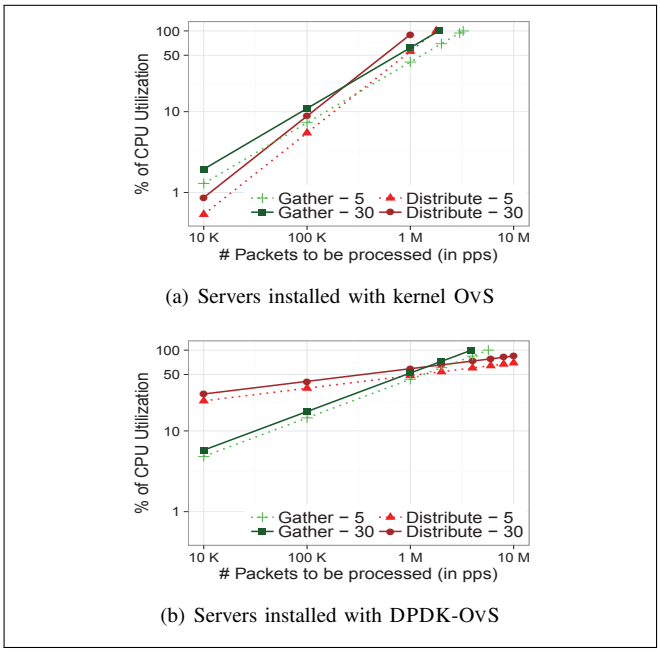
(a) 10Kpps and 50Kpps. (b) 100Kpps and 200Kpps (c) 1Mpps and 1.5Mpps

Fig. 8: Cpu-cost ranging over different service chain length ($\varphi|^n$), while receiving traffic generated in 1500Bytes packets, for both placement functions on servers that are installed with kernel OvS.



(a) 10Kpps and 50Kpps (b) 1Mpps and 1.5Mpps (c) 3Mpps and 6Mpps

Fig. 9: Cpu-cost ranging over different service chain length ($\varphi|^n$), while receiving traffic generated in 1500Bytes packets, for both placement functions on servers that are installed with DPDK-OvS.



(a) Servers installed with kernel OvS

(b) Servers installed with DPDK-OvS

Fig. 10: Cpu-cost ranging over different packet processing requirements ($\varphi|^p$) 1500Bytes per packet, for both placement functions

Figures 10(a) and 10(b) depicts the CPU cost for both placement functions when increasing the required number of packet to process (1500Bytes per packet) on servers that are installed with kernel OvS (Figure 10(a)) and DPDK-OvS (Figure 10(b)). In these graphs we focus on the definition of a feasible cpu-cost function. Again, we normalize the cpu-cost values (i.e., value 100% representes all CPU-core on all servers), and scale the amount of required packets to process, in order to illustrate the bounds. All values presented in the graphs are in log scale. The graphs reaffirm the results discussed in Figures 8 and 9, that is, deciding the appropriate placement strategy depends on the required network traffic demand to process.

Recall the definition of feasible cpu-cost function from Section II: a cpu-cost function is feasible if there are enough resources to implement it. In the results presented in Figures 8, 9, and in Figure 10, the value 100% indicates that we have reached the processing bound and we cannot process more packets. We can observe that the infeasibility point is reached differently in each deployment strategy. For instance, the cpu-cost function of the distribute placement $\mathcal{C}_{db}$ reaches its infeasibility point faster than the gather placement $\mathcal{C}_{gb}$ when using OvS in Kernel model (Figure 10(a)).

V. Related Work

Network function placement. A proliferating field of interest in NFV is VNF placement [10]–[12] and chaining [13]–[17] strategies. The efficient orchestration of VNFs and its routing (or chaining) plays a crucial role in the performance of deployed network services. In this regard, [12] focus on where to place VNFs and how to assign flows to them. In turn, [14], [15] continue and focus on joint optimization of VNF placement and chaining. Specifically, they defined optimized placement functions which take into account how VNFs are interconnected. However, all mentioned lines of work have neglected the cost of software switching and its limitations. In general, those works have focused on minimizing arbitrary cost functions. Therefore, the usage of such models in real NFV deployments might either lead to infeasible solutions or suffer form high penalties on the expected performance.

Software switching and IO acceleration. Many acceleration technologies were developed over the years ([18]–[21]). Accelerating packet processing enables applications such as switching ([6], [22]–[25]) and network stack ([26]) to process packets fast and efficiently. However unlike these technologies, our goal is not to delivers the best performance but rather to evaluate the performance and estimate its cost.

Middleboxes and Traffic steering. Both academia and industry show interest in virtualizing network appliances through a programmable, flexible and scalable architectures ([27]–[30]). Traffic steering is an essential building block for enabling flexible NFV deployment. SDN is a complementary technology that enables the dynamic traffic steering between middleboxes and commodity servers ([31]–[35]). However, most of the recent literature on SDN has analyzed inter-server traffic, rather than intra-server traffic – that is the focus of this work.

VI. Conclusion and Future Work

Software switching is a key component that enables the communication between VNFs. in any cloud based infrastructure. The rigid network requirements introduced by network function virtualization (i.e. high throughput and low latency) makes it a crucial component in that paradigm.

In this paper, we conduct an extensive and in-depth evaluation, measuring the performance and analyzing the impact of deploying service chains on a real NFV-based infrastructure. We provide insights on how throughput, packet processing and CPU consumption behave when scaling up service chaining deployments. Furthermore, we develop a generalized cost function that accurately captures the CPU cost of software switching in this setting.

We plan to incorporate our work in OpenStack and other orchestration tools (e.g. kubernetes [36], swarm [37]) in order to scale our evaluations, and analyze sequence of real NFV scheduling requests. Another future direction we plan to explore is the analysis and measurement of non-linear and sophisticated service chains.

Understanding the costs and the limitations of software switching in NFV environments is a key ingredient in the ability to design efficient solutions for VNF management and orchestration – possibly leading to lower operational costs. Thus, a natural extension of this work is to develop cost-efficient service chain deployment schemes based on the devised CPU-cost function.

REFERENCES

[1] J. Martins, M. Ahmed, C. Raiciu, V. A. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, 2014, pp. 459–473.

[2] ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV), "Network functions virtualisation," Available: http://www.etsi.org/technologies-clusters/technologies/nfv.

[3] P. Quinn and T. Nadeau, "Problem Statement for Service Function Chaining," Internet Requests for Comments, RFC Editor, RFC 7498, July 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7498.txt

[4] "Openstack," http://www.openstack.org/, accessed: 04-19-2016.

[5] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Proceedings of HotNets*, Oct. 2009.

[6] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130.

[7] "Open vswitch," http://www.openvswitch.org/, accessed: 06-09-2016.

[8] Intel, "Intel open network platform release 2.1: Performance test report," Internet Engineering Task Force, Mar. 2016, Available: https://01.org/packet-processing/intel®-onp.

[9] "Sockperf," https://github.com/Mellanox/sockperf/, accessed: 04-19-2016.

[10] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[11] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, Oct 2015, pp. 255–260.

[12] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1346–1354.

[13] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 7–13.

[14] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 98–106.

[15] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, ser. CNSM '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 50–56.

[16] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 89–97.

[17] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, April 2015, pp. 1–9.

[18] "Infiniband trade association," http://http://www.infinibandta.org/, accessed: 04-19-2016.

[19] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi, "On multi—gigabit packet capturing with multi—core commodity hardware," in *Proceedings of the 13th International Conference on Passive and Active Measurement*, ser. PAM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 64–73.

[20] L. Deri, "Direct NIC Access," Available: http://www.ntop.org/products/packet-capture/pf_ring.

[21] L. Rizzo, "Netmap: A novel framework for fast packet i/o," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–21.

[22] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 61–72.

[23] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G. Andersen, "Scalable, high performance ethernet forwarding with cuckooswitch," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 97–108.

[24] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 459–473.

[25] J. Hwang, K. K. Ramakrishnan, and T. Wood, "Netvm: High performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, March 2015.

[26] M. V. Bernal, I. Cerrato, F. Risso, and D. Verbeiren, "Transparent optimization of inter-virtual network function communication in open vswitch," in *5th IEEE International Conference on Cloud Networking, Cloudnet 2016, Pisa, Italy, October 3-5, 2016*, 2016, pp. 76–82. [Online]. Available: http://dx.doi.org/10.1109/CloudNet.2016.26

[27] A. Gember, R. Grandl, J. Khalid, and A. Akella, "Design and implementation of a framework for software-defined middlebox networking," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 467–468.

[28] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xomb: Extensible open middleboxes with commodity servers," in *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '12. New York, NY, USA: ACM, 2012, pp. 49–60.

[29] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 163–174.

[30] D. H. Anat Bremler-Barr, Yotam Harchol, "Openbox: A software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the ACM SIGCOMM 2016 Conference on SIGCOMM*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 1–15.

[31] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using sdn," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–38, Aug. 2013.

[32] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "Steering: A software-defined networking for inline service chaining," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

[33] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 543–546.

[34] G. W. Adiseshu Hari, T. V. Lakshman, "Path switching: Reduced-state flow handling using path information," in *Proceedings of the 11th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15. New York, NY, USA: ACM, 2015, pp. 1–7.

[35] J. R. A. S. J. R. Nanxi Kang, Monia Ghobadi, "Efficient traffic splitting on commodity switches," in *Proceedings of the 11th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15. New York, NY, USA: ACM, 2015, pp. 1–13.

[36] "Kubernetes," http://kubernetes.io/, accessed: 04-19-2016.

[37] "Swarm," https://docs.docker.com/swarm/, accessed: 04-19-2016.