Considerations on Deploying High-Performance Container-based NFV

DeokGi Hong*, Jaemin Shin*, Shinae Woo, Sue Moon
School of Computing, KAIST
{deokgi.hong,jaemin.shin}@kaist.ac.kr,shinae@an.kaist.ac.kr,sbmoon@kaist.edu

ABSTRACT

Over the last few years, the idea of network function virtualization (NFV) has become widespread, shifting the workload of hardware-based middleboxes onto applications on commodity hardware. Virtual machine is the common building block of the NFV platform, but its overhead on running additional OSes is a major drawback. With smaller overhead than VMs, containers are tested as the NFV platform, but their packet processing throughput is yet unsatisfactory. ($\approx$ 1.1Mpps).

In this paper, we focus on alternative performance evaluation of container-based NFV. We setup an evaluation system that forwards generated packets to container which returns back the network traffic to host and packet generator. As a result, we construct a container image that processes millions of packets per second (2.6Mpps) with low-latency ($\leq 25\mu s$) while keeping its size to the minimum (7.8MB). In addition, chaining of the container-based NF application shows that it preserves the high-performance (process over a million packets per seconds with latency $\approx 100\mu s$) while adding up to 5 containers into the chain. Our evaluation system of lightweight, high-performance NFV requires no kernel modification nor hardware acceleration. To this end, we utilize BESS as a virtual switch for containers. As the main form of our work is not in the application development, we take full advantage of Click for network functions implementation in our evaluation.

CCS CONCEPTS

• **Networks → Network performance evaluation**;

KEYWORDS

NFV, Container-based Virtualization

ACM Reference format:
DeokGi Hong, Jaemin Shin, Shinae Woo, Sue Moon. 2017. Considerations on Deploying High-Performance Container-based NFV. In *Proceedings of CAN '17: Cloud-Assisted Networking Workshop, Incheon, Republic of Korea, December 12, 2017 (CAN '17)*, 6 pages.
https://doi.org/10.1145/3155921.3155925

CAN '17, December 12, 2017, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5423-3/17/12…$15.00
https://doi.org/10.1145/3155921.3155925

1 INTRODUCTION

The recent introduction of Network Function Virtualization (NFV) has initiated the flexible use of middleboxes, solving significant weaknesses that hardware-based middleboxes had. Hardware-based middleboxes are expensive to deploy and hard to adjust the configuration subject to change in demand. NFV has transferred middleboxes on dedicated hardware to software running on general commodity hardware. For complete replacement, multi-tenancy and satisfactory performance are necessary to NFV. Virtual Machine (VM) has been the common building block for NFV platforms due to its strong isolation between instances. Although VM's default performance in packet processing is mediocre ($\approx$ 0.5Mpps) [7], but it achieves serviceable performance via hardware acceleration (e.g. SR-IOV) and packet path optimization (e.g. PCI passthrough).

Despite such useful features of VM in building NFV, VM has a few weaknesses in practice; the overhead of running a guest OS on a hypervisor for each instance is not negligible in both resource management and deployment effort. The size of each VM image containing guest OS is heavy (at least hundreds of megabytes) and the startup time of a single instance is lengthy (mostly takes few minutes). In addition, running hypervisor and guest OS requires excessive use of resources (CPU and memory) for a single network function. Due to these innate characteristics of VM, the number of VM deployment is limited [15]. Likewise, the slow startup time of VM also raises a serious concern if recovery of critical applications such as firewalls takes a few minutes.

Meanwhile, Linux Containers (LXC) has enabled fast, lightweight deployment of applications. The container is an OS-level virtualization approach of running Linux systems on the single kernel. Since containers share kernel of a host OS instead of running hypervisor and guest OS (Figure 1), containers are more lightweight than VMs, with fast startup time ($\leq$ few seconds). Moreover, containers enable more efficient resource management in operation than VMs which allocate additional computing resource to guest OS. In consequence of these characteristics, containers are widely used for application deployment as cloud computing platforms provide container-based application deployment service such as Amazon EC2 Container Service [1], Azure Container Service [6] and Google Container Engine [5].

As a NFV platform, containers are alternative [13] for VMs because of its lightweight nature. Also, Anderson et al. [10] shows the potential for container-based NFV, however, this work does not evaluate the throughput.

There are two requirements of a NFV platform: multi-tenancy and satisfactory performance.

*Joint first authors

Multi-tenancy. NFV platforms which concurrently operate multiple network function should ensure isolation between instances to support multi-tenancy. By default, containers are designed to provide isolation between instances on the same host by adopting two components from Linux kernel, cgroups and namespaces. Both components ensure each container the isolation on resource allocation and individual authority to access divided file system. Still, share of host kernel between containers induces concern on certainty of container's isolation.

Performance. Acceptable packet processing performance in throughput and latency ($\lesssim 100\ \mu s$) is required for NFV to substitute hardware-based middleboxes. Container adopts Linux bridge by default network interface, which behaves as a virtual network switch. Using this bridge, the throughput is about one million packets per second ($\approx$ 1.1Mpps) [12]. SR-IOV, the NIC virtualization that creates virtual ports from NICs, is also applicable to container to reach higher performance. However, using SR-IOV creates dependency on the machine's hardware NICs.

In this paper, we focus on performance evaluation of alternative setup of container-based NFV using Berkeley Extensible Software Switch (BESS) [2, 14], Data Plane Development Kit (DPDK) [3] for high-performance packet I/O, and Click [16] for implementation of network functions. We discuss the choices of these techniques in Section 2. Since these techniques process packet only in the user space, we achieve satisfactory performance without kernel stack modification. By using these components, a container running Click applications serve 2.8Mpps of throughput with low latency below 40 μs, while being deployed in minimal size of 7.8MB. Furthermore, additional evaluation on chaining of the containerized Click application shows that it operates in high-performance (process over a million packets per seconds with latency $\approx 100\mu s$) while the chain extends up to 5 containers.

2 DESIGN CHOICES

To build flexibly applicable container-based NFV platform with high performance, we utilized the virtual switch and the network function with following techniques: BESS, DPDK, and Click. In this section, we provide an explanation on our system design in detail, why we have selected the components and how they operate.

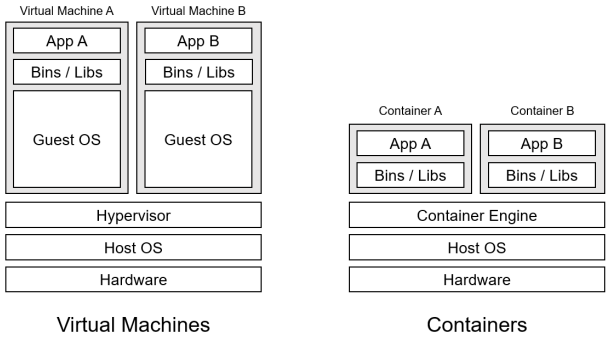


Figure 1: Infrastructure of VMs and Containers

2.1 Virtual Switch

In most cases of container usage, default Linux bridge of kernel is chosen as a virtual switch that relays network traffic between the host machine's NICs and containers. Nonetheless, using Linux bridge results in low performance due to data path through the kernel (Figure 2(a)). Different type of virtual switch is required for virtualized network function, since it demands higher performance than the default Linux bridge is capable of. As alternative solution, any other kind of virtual switch is applicable as long as it is supported on the Linux system such as BESS, OVS, macvlan, or even SR-IOV. Among these candidates, we have chosen BESS to address the performance issue.

BESS is a framework to build virtual switches. With the custom configuration, it is able to operate BESS as a switch, or even as the IPv4 router (More detailed explanation is provided in section 3: evaluation.) We included BESS in our design since BESS provides useful interfaces for packet processing with container while assuring high performance as network function. To provide high performance in throughput, BESS manipulates the data path for packet to bypass kernel by using DPDK, which supports high-performance packet processing on userspace. BESS provides two modules for hooking containers up out-of-box.

VPort. VPort is used for applications that needs kernel network stack such as Snort [9]. In case of using this interface, throughput cannot saturate the line-rate since kernel network stack is the bottleneck.

DPDK PMDPort. On the other hand, BESS directly transmits and receives packets with containers detouring the kernel. (Figure 2(b)) PMDPort creates a vhost-user backend socket in host, which a container creates virtio-user device using that backend. DPDK-enabled applications use this virtio-user [4] device and saturate the line-rate.

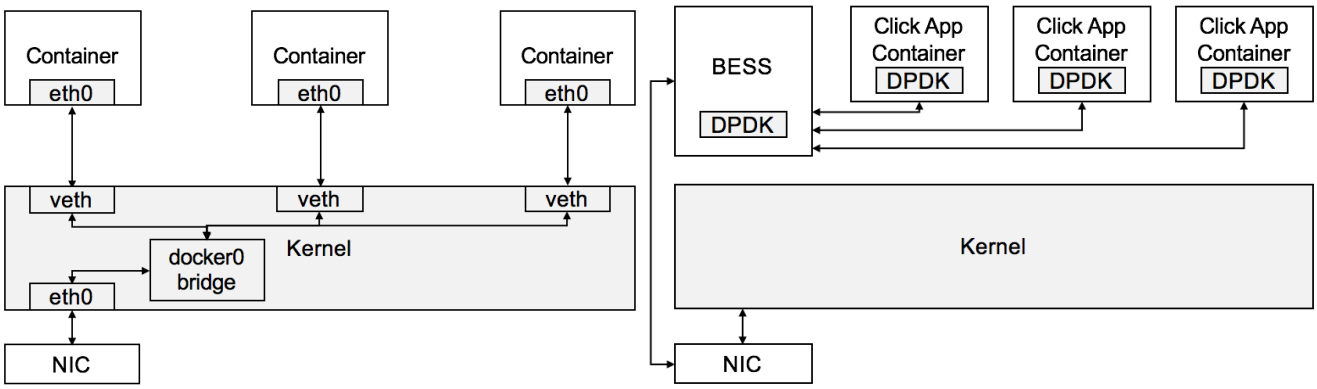
2.2 Network Functions

```
FROM scratch
COPY ./userlevel/click /click
CMD ["/click"]
```

Listing 1: Dockerfile for Click

To cover extensive use cases, we take Click to build network functions. It provides a bunch of re-usable elements for building various kinds of network functions such as load balancer, intrusion detection system, firewall, NAT, and IP router. Moreover, building such network function with Click requires only a simple effort on implementation. For example, building a standard-compliant router [8] requires only sixteen elements that Click provides. In addition, Click exploits DPDK for high performance packet processing.

In order to package Click as a Docker image, we utilize static compilation which Click provides as default. Listing 1 is the content of the Dockerfile of Click Docker image whose size is 7.8MB. To run a network function with this Click image, we add a Click configuration as a parameter before creating the container.



(a) Linux Bridge (b) BESS

Figure 2: Comparison on Packet datapath

3 EVALUATION

3.1 Test environment

The test server has two NUMA nodes and each node consists of Intel 12-core Xeon E5-2670 2.30GHz CPU, 64GB DDR4 memory and single-port Mellanox ConnectX-3 40GbE NIC. Packet generation and measurement (pktgen server) are conducted on an external server which has the same specification except the memory. The pktgen server has 32GB DDR4 memory for each node. Two servers are connected through a Mellanox SX1036 switch. For deterministic tests, we disable Hyper-Threading and power management features in BIOS.

Both servers run on unmodified Ubuntu 16.04.2 LTS with Linux kernel 4.4.0. We use Docker 17.03.1-ce for the container engine, BESS at '38192c5' for the virtual switch, Click at '7c872086', and DPDK 17.02 configured to enable Mellanox NICs support for user-space packet I/O.

3.2 Experiment Setup

To test and analyze the networking performance of container-based NFV, we designed three experiments; baseline performance measurement, application performance measurement, and NF (Network Function) chaining performance measurement. For performance measurement, we utilized two servers; pktgen server and test server. Pktgen server generates the network traffic and transmits it to test server, and measures the test server's networking performance as the traffic returns. Test server processes the network traffic as desired for each experiments. The size of measured packets are 64, 128, 256, 512, 1024, 1500 bytes.

Baseline performance is measured in order to define the performance capacity of the system which every packets from NICs are processed by BESS. Figure 3(a) illustrates the packet flow in the baseline performance measurement. The pktgen server transmits packets to the test server where BESS processes packets, which relays the packets back to the pktgen server. For this measurement, we configure BESS to use a 'MacSwap' module which swaps a packet source and destination address. The 'MacSwap' module is included in BESS as default.

Application performance implies the performance of container-based NFV. In this experiment, we measured the networking performance of the system running containerized Click application, EtherMirror.

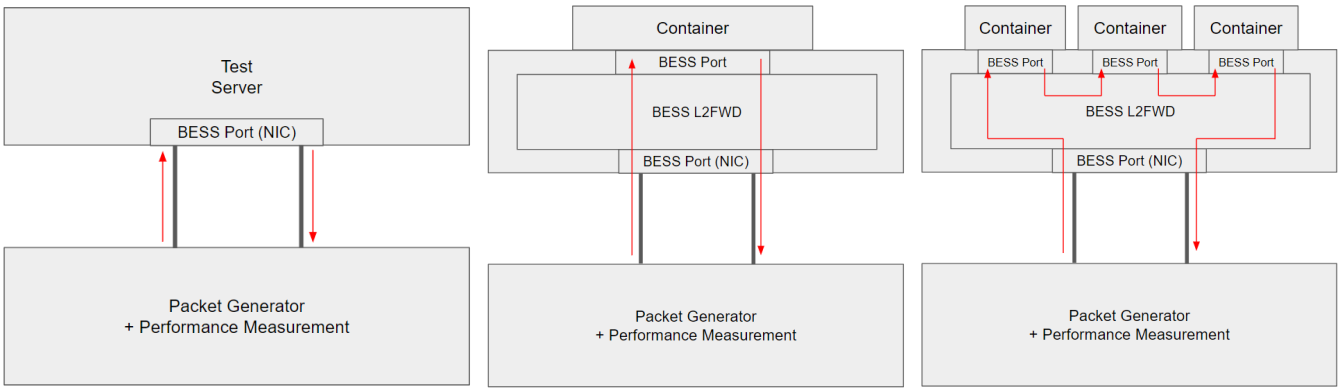
EtherMirror This application swaps the ingress packet's source and destination MAC address. Click provides this feature as the EtherMirror element, so we configure this application with following code:

```
FromDPDKDevice(0) -> EtherMirror -> ToDPDKDevice(0)
```

Listing 2: Click configuration for EtherMirror

In terms of testing the performance of container-based NFV, EtherMirror is chosen in our system to evaluate the network processing capacity of container. In addition, EtherMirror is chosen to represent the performance of Click application, which builds various network function as software.

Figure 3(b) illustrates the packet flow of the experiment. The pktgen server transmits packets to the test server, where the virtual switch relays the packets to the container running EtherMirror application. Eventually, the application on container forwards the packets all the way back to the pktgen server. As a virtual switch in the system, we configure BESS as a Layer 2 switch with the following configuration:



(a) BESS baseline (b) Single container IO (c) Chaining of multiple NFs

Figure 3: Experiment Environments

```
l2fib = L2Forward()
# default gate
entries=[{'addr':'e4:1d:2d:17:58:70', 'gate': 1}]
for i in range(num_ports):
n = i + 2
mac = '56:48:4f:53:54:{:02x}'.format(n)
entries.append({'addr': mac, 'gate': n})
l2fib.add(entries=entries)
ports = []
for num in range(num_ports):
ports.append(PMDPort(name='sock{}'.format(num),
    vdev='eth_vhost{0},iface=/tmp/sock{0}.sock,
    queues=1'.format(num)))
p = PMDPort(pci='03:00.0')
PortInc(port=p.name) -> l2fib
for j in range(num_ports):
n = j + 2
l2fib:n -> PortOut(port=ports[j].name)
PortInc(port=ports[j].name) -> l2fib
l2fib:1 -> PortOut(port=p.name)
```

Listing 3: BESS configuration for a Layer 2 switch

In this configuration, incoming packets are dispatched to the container by using BESS's L2Forward module which looks up a packet's MAC address and forwards the packet to output gate. When BESS initializes with this configuration, vhost-user backed ports are created and registered to the forwarding table with their MAC addresses.

For application performance measurement, we instantiate one containerized Click application which binds to a single core. The network interface is provided as a single DPDK virtual device which uses a virtio-user driver with the vhost-user backend created by BESS.

As real application of NF generally requires network traffic to go across multiple, chained NFs in sequence [17], we measured the performance of chained container-based NFVs. The main focus of this experiment is on measurement of overhead originated from the act of container chaining itself, to examine if the container-based NFV platform maintains high-performance as the number of chained containers increases. In the experiment, number of chained containers scaled from 1 to 5, each of containers on respective single core running same Click application, EtherEncap.

EtherEncap This application rewrites the ingress packet's source and destination MAC address as user inputs both addresses as parameter, while encapsulating the packet's ethernet header (0x0800 is used as IPv4 header). Due to its encapsulation functionality, EtherEncap requires the beforehand use of Click application, Strip, which strips the ethernet header of the packet. We configure the chained container's Click application with following code:

```
FromDPDKDevice(0) -> Strip(14) -> EtherEncap(0x0800, desired src,
    desired dst) -> ToDPDKDevice(0)
```

Same as application performance measurement, we configure BESS as a Layer 2 switch and provide single vhost-user backed port in connection for each single container. In order to forward packets from container to another port paired with container in chain, EtherEncap is chosen to rewrite the destination MAC address as we desire. With such functionality of EtherEncap, it is able to forward packets to next container in the chain or back to the pktgen server to end the chain. As configured, Figure 3(c) illustrates the packet flow of the NF chaining experiment.

3.3 Baseline Performance

A black bar in Figure 4 shows the result of the throughput with different packet size. BESS processes 5 Mpps in case of minimum-size (64 byte) packet, which is lower than reported performance of BESS. BESS also shows saturation of 40G link with 168-byte packets using single core. We suspect that the low throughput originates from the Mellanox NIC. With the bigger packet size, BESS fills up the line rate up to 38Gbit/s.

We also measure the end-to-end latency that BESS incurs in the same manner. The pktgen server generates 64-byte packets at 0.5Mpps which is a loss-free rate. The mean latency is 9.3 usec (stddev = 0.4) regardless of packet size.

3.4 Application Performance

A gray bar in Figure 4 reports the throughput of the containerized EtherMirror application. Our container system with Click application processes 2.8Mpps at best. The gap between the baseline and the application decreases as the packet size increases.

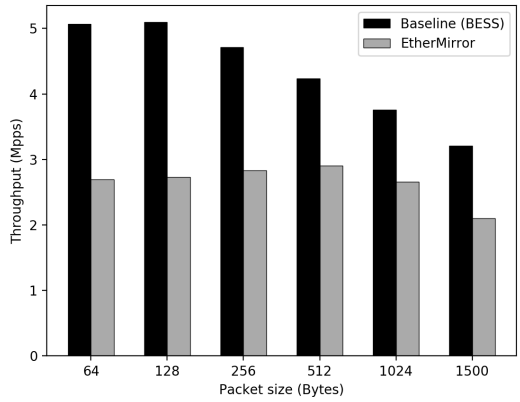


Figure 4: Throughput of baseline system and Click network function, EtherMirror running in a single container on test server, measured with packets of 6 different sizes

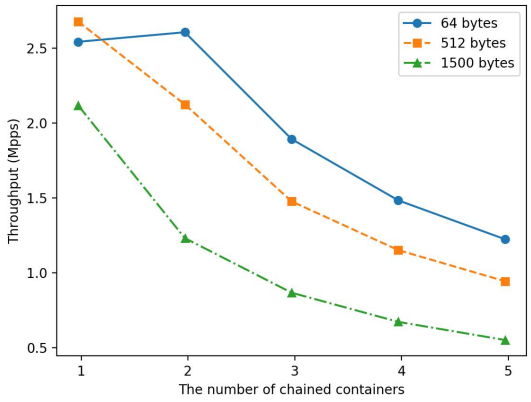


Figure 6: Throughput of chained containers running Click network function, measured with packets of size 64 bytes, 512 bytes and 1500 bytes

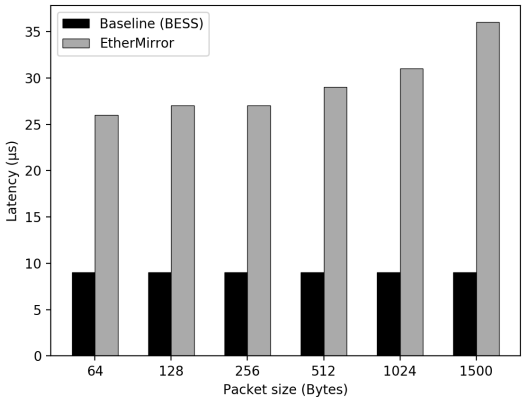


Figure 5: End-to-end latency of baseline system and Click network function, EtherMirror running in a single container on test server, measured with packets of 6 different sizes

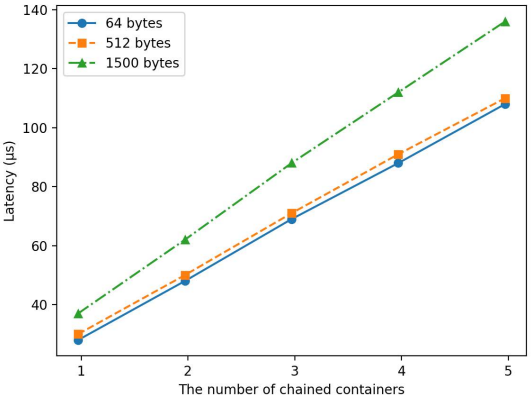


Figure 7: End-to-end latency of chained containers running Click network function, measured with packets of size 64 bytes, 512 bytes and 1500 bytes

The end-to-end latency of the system is shown in Figure 5. Comparing with the baseline latency, latency of this experiment has 16 - 26 μs delay. The delay increases as the packet size increases, contrary to the baseline end-to-end latency which is constant regardless of the packet size.

Overall, the performance of Click application is lower than the baseline performance. This is the same when it runs in bare-metal environment. This low performance originates from the Click application, not from the container platform. We could improve the Click application performance various methods that fastclick [11] introduced, but we left it as a future work.

3.5 NF Chain Performance

The throughput performance of chained containers which runs Click application is demonstrated in Figure 6. The throughput (packets per second) decreases as the number of chained containers increases, except for the case of 64 byte packet which the throughput increases as the number of chained containers increases from 1 to 2. The result shows that system of chained containers running Click application is able to process over a million minimum-size and 512 bytes packets per second while chaining up to 5 containers.

Figure 7 shows the end-to-end latency of the system of chained containers. The end-to-end latency of the system increases as the number of chained containers increases. The rate of increase in latency varies by packet size, however, they increases linearly with $\approx 20\mu$s per addition of one container to the chain. The result reports

that the system is able to process packets at ≈ 100 μs latency while chaining up to 5 containers.

4 RELATED WORK

High-performance NFV enablers

ClickOS [15] is the Xen-based NFV deployment platform. It optimizes Xen's network stack with netmap, VALE switch and modified network interfaces. Also, it builds a tiny OS image that is specialized to run Click elements, so that its startup time is fast in about 30ms while minimizing the image size at 5MB. Our work is highly inspired by ClickOS in a way that containers are also available of being the similar platform for network functions. We utilize Click applications using BESS and DPDK for user-space packet processing, providing the comparable isolation level on top of containers.

Recently, DPDK [3] introduced a virtio-user driver [4] for container networking between DPDK-enabled applications. While container's networking performance through Linux bridge in kernel is low, virtio-user devices with vhost-user backend enable high performance packet I/O bypassing the Linux kernel. In addition, the virtio-user devices are highly utilizable because they are available of being attached to any DPDK applications. We apply this user space networking to containerized Click applications for high-performance packet processing.

NF Chainings

Anderson et al. [10] measured the performance of containerized network functions with different switches such as OVS, macvlan, SR-IOV, etc. In addition, they evaluated the performance of NFV service chain which NF instances process packets in sequence. This work, however, mainly focused on measuring the end-to-end latency and did not present the throughput results. In our work, we measured not only the end-to-end latency, but also the throughput of container-based NF instances in chain.

5 CONCLUSION

In this paper, we create a small Click container image for running network functions and show that containerized Click application process millions of packets per second with the small delay less than 40μs while being deployed in minimal size of 7.8MB. BESS as a virtual switch and DPDK's user-space networking enable high performance container networking. Moreover, our evaluation shows that chaining of containerized Click application preserves the high-performance (process over a million packets per seconds with latency ≈ 100μs) while including at most 5 containers to the chain.

ACKNOWLEDGEMENTS

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the National Program for Excellence in SW supervised by the IITP(Institute for Information & communications Technology Promotion). (2016-0-00018)

REFERENCES

[1] Amazon EC2 Container Service. https://aws.amazon.com/ecs/. (accessed June 15, 2017).
[2] BESS (Berkeley Extensible Software Switch). https://github.com/NetSys/bess. (accessed June 15, 2017).
[3] DPDK (Data Plane Development Kit). http://dpdk.org. (accessed June 15, 2017).
[4] DPDK Virtio_user for Container Networking. http://dpdk.org/doc/guides/howto/virtio_user_for_container_networking.html. (accessed June 15, 2017).
[5] Google Cloud Platform Container Engine. https://cloud.google.com/container-engine/. (accessed June 15, 2017).
[6] Microsoft Azure Container Service. https://azure.microsoft.com/en-us/services/container-service/. (accessed June 15, 2017).
[7] Network Throughput in a Virtual Infrastructure. http://www.vmware.com/pdf/esx_network_planning.pdf. (accessed June 15, 2017).
[8] Requirements for IP Version 4 Routers. https://tools.ietf.org/html/rfc1812. (accessed June 15, 2017).
[9] Snort - Network Intrusion Detection & Prevention System. https://www.snort.org. (accessed June 15, 2017).
[10] Jason W. Anderson, Hongxin Hu, Udit Agarwal, Craig Lowery, Hongda Li, and Amy W. Apon. 2016. Performance considerations of network functions virtualization using containers. *2016 International Conference on Computing, Networking and Communications (ICNC)* (2016), 1–7.
[11] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast userspace packet processing. *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (2015), 5–16.
[12] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems.
[13] ETSI. Network Functions Virtualisation (NFV); Infrastructure; Hypervisor Domain. http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/004/01.01.01_60/gs_NFV-INF004v010101p.pdf. (accessed June 15, 2017).
[14] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. SoftNIC: A Software NIC to Augment Hardware.
[15] João Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Andrei Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the Art of Network Function Virtualization. In *NSDI*.
[16] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. 1999. The Click modular router. *ACM Trans. Comput. Syst.* 18 (1999), 263–297.
[17] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. 2017. NFP: Enabling Network Function Parallelism in NFV. In *ACM SIGCOMM*.