Performance Considerations of Network Functions Virtualization using Containers

Jason Anderson
School of Computing
Clemson University
Clemson, South Carolina
jwa2@clemson.edu

Udit Agarwal
School of Computing
Clemson University
Clemson, South Carolina
uagarwa@clemson.edu

Hongda Li
School of Computing
Clemson University
Clemson, South Carolina
hongdal@clemson.edu

Hongxin Hu
School of Computing
Clemson University
Clemson, South Carolina
hongxih@clemson.edu

Craig Lowery
Dell Corporation
Round Rock, Texas
Craig.Lowery@software.dell.com

Amy Apon
School of Computing
Clemson University
Clemson, South Carolina
aapon@clemson.edu

***Abstract*—The network performance of virtual machines plays a critical role in Network Functions Virtualization (NFV), and several technologies have been developed to address hardware-level virtualization shortcomings. Recent advances in operating system level virtualization and deployment platforms such as Docker have made containers an ideal candidate for high performance application encapsulation and deployment. However, Docker and other solutions typically use lower-performing networking mechanisms. In this paper, we explore the feasibility of using technologies designed to accelerate virtual machine networking with containers, in addition to quantifying the network performance of container-based VNFs compared to the state-of-the-art virtual machine solutions. Our results show that containerized applications can provide lower latency and delay variation, and can take advantage of high performance networking technologies previously only used for hardware virtualization.**

I. Introduction

In the past few years, Network Functions Virtualization (NFV) has positioned itself as a paradigm shift in enterprise networking. All of the benefits of virtualization - server consolidation, higher availability, and scalability [1] can be brought to the world of network middleboxes [2]. Network functions such as firewalls, routing, and intrusion detection systems, which are typically implemented as specialized hardware appliances, can instead be deployed as virtualized network functions (VNFs) on cost efficient commercial off-the-shelf hardware.

While traditional virtual machines (VMs) have so far been the target of NFV implementations, there are a number of challenges that must be addressed as NFV evolves into a mainstream technology:

1) Hardware virtualization incurs significant performance and efficiency costs [3].
2) Heterogeneous packaging and virtualization platforms may result in fragmented distribution and complex orchestration.
3) VM images can be large, resulting in significant time to deploy and migrate statefully between hosts.
4) Network I/O, which is of critical importance to NFV, can also suffer in many configurations.

Operating system-level virtualization (i.e., *containers*) is a relatively new technology which allows applications to run as sandboxed user-space instances on the host machine with isolation similar to hardware virtualization. Container packaging and deployment platforms such as Docker[1] simplify using containers to run virtualized services by packaging applications with a customized view of their runtime environment. Container ecosystems like Docker have the potential to address some of the above challenges:

1) Since applications in containers run on the host OS without hardware indirection, they can run more efficiently than their VM-based counterparts [4] and allow higher application density on a host [5].
2) Docker's novel packaging can remove some of the variability in hosting requirements that a VNF may express. Projects like the Open Container Initiative[2] aim to standardize container formats and make them even more platform agnostic.
3) Containers do not require packaging the operating system in their image, and as such generally consume much less disk space than comparable VMs, decreasing time to deploy and migrate. Live migration of stateful containers has been explored [6] and implemented in real world systems like Flocker[3].

Docker seems to be a natural fit for NFV. However, one challenge that is not sufficiently addressed by containers is network I/O. Containers are typically used to provide isolation for services that communicate using one or more network sockets bound to a port on the host. Traffic is handled by the host's network stack using a software switch such as the Linux bridge, which can incur performance cost and variation. While many services typically deployed in containers are not bounded

[1]http://www.docker.com
[2]http://www.opencontainers.org
[3]http://clusterhq.com/flocker

by network performance, most use cases for NFV have strict requirements for network throughput and delay [7] that can be difficult to guarantee with traditional Linux networking.

Given the potential benefits in containers for VNFs, we should better understand their shortcomings and devise ways to overcome them, rather being dismissive of them. Full hypervisor virtualization suffered from similar criticisms in its infancy and research into these problems resulted in hardware and software innovations that greatly improved performance and security to the point that hypervisor virtualization is the default choice of infrastructure for most applications. The NFV concept validates the fact that hypervisor virtualization is believed sufficient to the task and there is no known reason yet that containers cannot advance in similar fashion to be suitable for the task.

Our research is the first in a line of investigations to better understand the real, rather than perceived networking issues in operating system level virtualization. It is the goal of this work to identify and quantify the factors that influence the packet delay and throughput of container-based applications and virtual machines, in the context of NFV service chains where VNF instances may exist on the same or multiple hosts on a network. We describe and report on controlled experiments devised to isolate these factors, and finally identify goals of future research in this area.

II. Background and Technology Selection

In traditional enterprise and telecommunications networks, network services such as routing, intrusion detection systems, and firewalls are typically performed by specialized hardware appliances situated in the data plane. With NFV, these services are instead implemented as software applications that run in virtual environments on standardized general purpose computing hardware. This gives huge benefits in flexibility and scalability, at the cost of lower operating efficiency through virtualization and the use of general purpose hardware. It also opens up network services to management and orchestration techniques that allow for unprecedented control, and removes much of the complexity and specialized knowledge required with traditional systems.

A. Virtualization

A traditional virtual machine, i.e. *hardware-level* or *hypervisor* virtualization, is an abstraction of physical hardware giving each VM a full server hardware stack including virtualized network adapters, storage and CPU. Virtualizing the entire hardware stack means that each VM needs a complete operating system for instantiation.

Operating system (OS) level virtualization – also known as *jails* or *containers* – is a method of isolating the view of the operating system for an application. On a Linux host, this is typically done using cgroups to limit resource usage, along with namespaces to isolate the user's view of process trees, networking, and filesystems. Since a separate kernel does not load for each user session, the overhead associated with multiple operating systems is not experienced, as shown in Fig. 1. A perfectly tuned container system can have as many as four to six times the number of server application instances as is possible using Xen or KVM on the same hardware [8]. OS-level virtualization has been used successfully in applications where maximum resource utilization efficiency is required, such as Linux-VServer in PlanetLab [9]. Among other benefits of containers, two of the most relevant in the context of NFV are the lower overhead and shorter software stack between the Network Interface Controller (NIC) and the application.

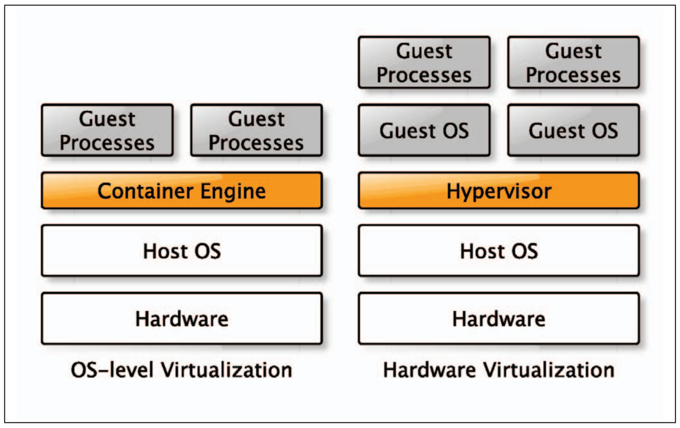

Fig. 1. Software layers of container-based (OS-level) and hypervisor-based (hardware-level) virtualization.

1) Docker: We selected Docker as a representative OS-level virtualization platform for a number of reasons. First, it allows us to build layered container images to ensure that the host OS meets the dependency requirements of the VNF, as well as distribute these images though public or private repositories. Second, while our configuration uses the default libcontainer library for access to the host's virtualization features, Docker is capable of using other libraries such as libvirt and LXC should they be more suitable to the use case. These benefits and others place Docker as a container platform to be particularly well suited for use in an NFV context. OpenStack[4], a fundamental tool in NFV deployments and the foundation of the OPNFV project[5], already supports Docker as a hypervisor driver[6]. Other software, such as Shipyard[7], can control clouds of hosts running Docker, making it even more suitable as a target for cloud-based NFV.

2) Xen: We use Xen [1] to represent hypervisor virtualization. Xen is considered one of the most mature and efficient virtualization solutions [10], and has been well studied in the context of network performance [11]. Second, Xen is supported by OpenStack and identified as one of the primary infrastructure targets for OPNFV. Finally, Xen VM-based VNFs have been explored in previous work [7], which validates it as a comparable technology.

B. Networking Technologies

Several networking technologies were identified as both commonly available on modern enterprise software and hardware systems, and potentially beneficial over the default subsystems available for routing network traffic between VMs or

[4]http://www.openstack.org
[5]http://www.opnfv.org
[6]http://wiki.openstack.org/wiki/Docker
[7]http://shipyard-project.com

containers on Linux. We classify these technologies as either a software switch, which is a software construct independent of a physical network device, or device virtualization, which is applied to a physical interface.

1) Network Device Virtualization: Macvlan allows the abstraction of a single network interface into multiple network interfaces with different hardware addresses assigned to them. While operating in bridge mode, macvlan acts as a lightweight layer 2 software switch, using the frame's MAC address for distribution [12].

Single Root I/O Virtualization (SR-IOV) is a specification that allows a physical PCIe device to present itself to the OS as multiple separate PCIe devices called virtual functions(VFs), each with dedicated queues for transmitting and receiving packets. In the virtualized environment each virtual machine is directly assigned a VF by the hypervisor. When a packet arrives at the NIC, it is classified by a hardware L2 sorter and then sent to the pool assigned to the packet's hardware address or VLAN tag. It is then transferred via direct memory access to the guest OS memory space. Since data is transferred directly to and from a VM without the intervention of a software switch, SR-IOV removes the CPU from the process of moving data to and from a virtual machine.

2) Software Switches: The Linux bridge is a layer 2 software switch included with the Linux kernel, and is commonly used to connect two Ethernets together. In virtualization, the bridge is typically used to switch packets between VMs and to external hosts over a physical interface. Performance compared to a hardware switch is dependent on system load [13].

Open vSwitch[8] (OVS) is an open source OpenFlow [14] capable virtual switch that is commonly used in a virtualized environment to interconnect VMs within a host. In flow based networks, a network node has capabilities that identify a flow from Open vSwitch Database management protocol and execute appropriate actions for the matched flow. Two components of OVS are user-mode daemon and kernel forwarding module. Whenever the first packet of a new flow passes through the OVS kernel module, it is sent to OVS daemon which evaluates the OpenFlow rules, accepts or drops the packet and installs the corresponding per-flow forwarding rule into the kernel module.

III. Performance Comparison

Since chains of VNFs are meant to replace fast, high throughput hardware middleboxes, the maximum throughput, latency cost, and delay variation of the service chain is of primary importance. Throughput can be addressed to an extent by horizontal scaling, but there will always be a minimum delay cost of the chain even with minimal load. Previous work has found that virtualization can introduce throughput instability and abnormal delay variations to the network traffic [15].

All our experiments were conducted on bare metal instances in CloudLab [16]. Each physical machine was a Dell C8220 server, with dual Intel Xeon E5-2660v2 10-core 2.20Ghz CPUs, 256GB ECC RAM, and Intel 82599SE 2-port 10Gbe network interface controller (NIC). In each experiment, machines were co-located on the same rack and connected

[8]http://openvswitch.org

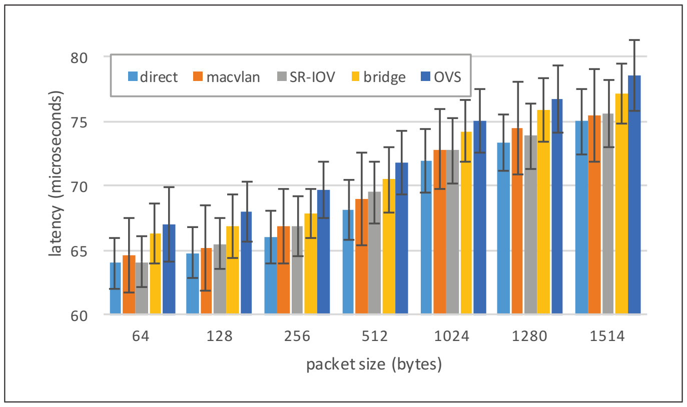

Fig. 2. Latency and standard deviation of various-sized Ethernet packets from an external host to processes in Docker containers, using different networking technologies.

by two networks: a 1Gbps control network, and a 10Gbps experiment network linked to a Dell Force10 S6000 switch.

All machines ran Ubuntu 14.04 LTS with the 3.13.0-57 low latency Linux kernel. In all experiments, task pinning and kernel scheduling exclusion were used to ensure that kernel threads, hardware interrupt servicing threads, and user threads were run on separate cores within the same NUMA node. While this approach explicitly disallows L1 and L2 cache reuse between threads, the lesser degree of context switching allows us to obtain more reproducible results.

In our experiment configurations, interfaces are added to containers using network namespaces. *SR-IOV* VFs are created by the OS after setting the `num_vfs` parameter of the NIC. *OVS* and *bridge* use virtual Ethernet device pairs assigned to the switch and the container. These device pairs, as well as *macvlan* subinterfaces, are created with the `ip` command. Interfaces are then moved into the container's network namespace, similar to the *direct* assignment of a physical interface to a container.

Packet send and receive timestamps were recorded for measurements of jitter and lateness, a method validated by [17] and [18], and collected using tcpdump[9]. Measurements of latency were conducted using Netperf 2.6.0[10]. Ethernet frame sizes of 64, 128, 256, 512, 1024, 1280, and 1514 bytes were chosen according to the standard network device benchmarking methodology established by RFC 2544 [19].

A. Baseline Performance

To establish a set of baseline performance comparisons, we evaluated each of the networking mechanisms for connecting processes in three environments – *Docker containers*, *Xen virtual machines*, and *running natively on the host* – and using three metrics: *latency*, *jitter*, and *efficiency*. In each test, UDP packets were sent from a client running on an external host to a server running on the virtualization host.

1) Latency: We first evaluated the networking technologies using latency to compare the costs of their different

[9]http://www.tcpdump.org
[10]http://www.netperf.org

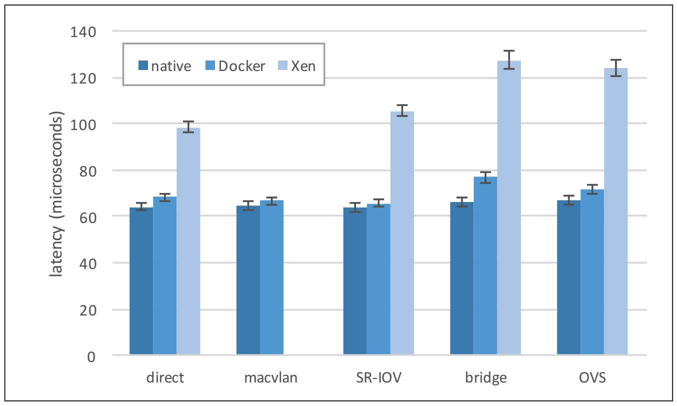

Fig. 3. Latency and standard deviation of 64-byte Ethernet packets from an external host to processes running natively, in Docker containers, and in Xen virtual machines.

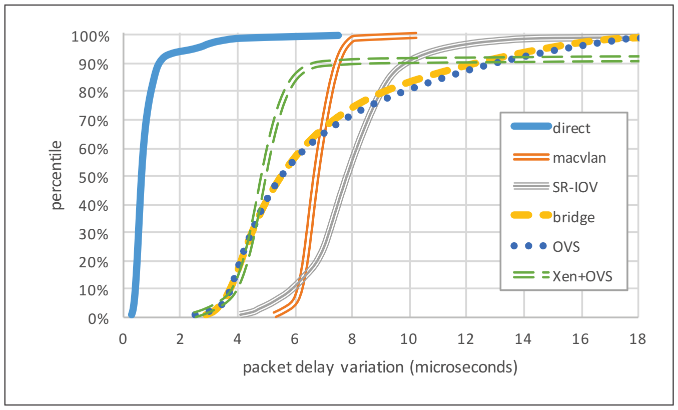

Fig. 4. Packet delay variation (jitter) of 64-byte Ethernet packets received by native processes, using different network technologies. Measurements using Xen virtual machines and OVS are included for comparison.

software stacks. Fig. 2 illustrates the results. We found that for each packet size, device virtualization mechanisms such as macvlan and SR-IOV incurred less processing delay than software switches. On average, the Linux bridge and OVS increased latency 3.2% and 4.9% over the native network stack, respectively, while macvlan and SR-IOV increased the latency by 1.1% and 1.0%.

We then compared latency cost of sending packets to Docker containers, Xen virtual machines, and host native processes. The results in Fig. 3 show that with each networking technology, containerized applications carry an additional latency cost (2.6%-16.1%) compared to native applications, but less of a penalty than the equivalent Xen VMs (53.9%-92.3%). Our findings of higher mean latency with Xen agree with those of other researchers [18].

2) Jitter: Next, we measured the delay variation (i.e., *jitter*) of packets sent from an external host to a receiver on the virtualization host using each of the network technologies. Ethernet frames of 64 bytes were generated at a constant bitrate of 130Mb/s, and samples were taken excluding the head and tail of the stream. Jitter was calculated according to the standard set forth in [20], designed to account for clock skew between the sender and receiver, and defined as:

$$D(i,j) = (R_j - R_i) - (S_j - S_i)$$

$$J(i) = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16}$$

where $D(i,j)$ is the difference between packet spacing at the sender and receiver, S and R are the sending and arrival timestamps of a series of packets I, and $J(i)$ is a continuous calculation with the gain parameter $1/16$ for noise reduction.

As maximum delay is also an important metric in NFV as late packets influence packet loss ratio[21], we also measured the lateness of packets relative to the minimum observed delay time. Fig. 4 illustrates the results, which are detailed in Tables I and II. In our experiments, we found macvlan to have the most stable jitter and lateness, while the Linux bridge and OVS experienced frequent delays in the milliseconds, despite a low mean variation. Included are results of a Xen VM receiving packets through OVS, which experienced significantly higher

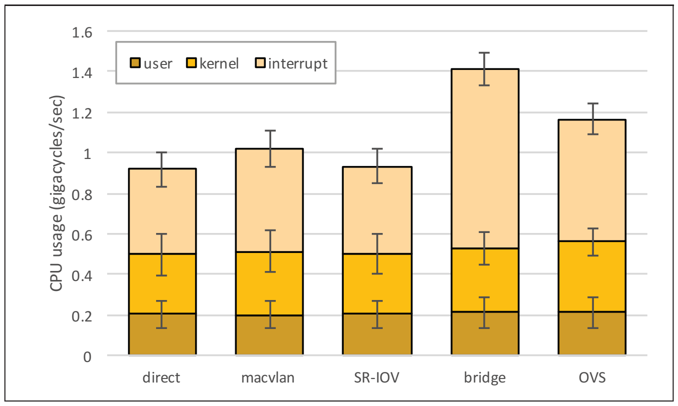

Fig. 5. Computational efficiency and standard deviation of networking mechanisms forwarding 64-byte Ethernet packets from an external host to a Docker container, classified into user processes, kernel processes, and interrupt servicing.

variation than a native process served by OVS, along with a many packets (0.61%) arriving out of order.

TABLE I. PACKET DELAY VARIATION (μsec)

	direct	macvlan	SR-IOV	bridge	OVS	Xen+OVS
n	50000					
min	0.17	2.57	3.00	1.60	1.35	1.42
max	7.50	10.25	27.97	20.71	24.47	18644.02
$\bar{x}$	0.87	6.79	7.96	6.76	6.96	265.36
s	0.69	0.55	1.87	3.43	3.71	1233.03

TABLE II. PACKET LATENESS (μsec)

	direct	macvlan	SR-IOV	bridge	OVS	Xen+OVS
n	50000					
max	214	114	391	3326	9727	72867
$\bar{x}$	41.40	12.22	53.83	1685.29	6068.86	782.85
s	40.49	5.88	43.23	1181.90	2289.98	6015.96

3) Efficiency: While latency is an important metric, it is not necessarily an indicator of the throughput of the system. We measured the efficiency of each networking mechanism by sending a controlled-rate stream of 64-byte packets (˜64Kpps) from an external host to a receiver within a Docker container, while observing the CPU usage as reported by the system. Fig. 5 shows the results according to user, hardware interrupt,

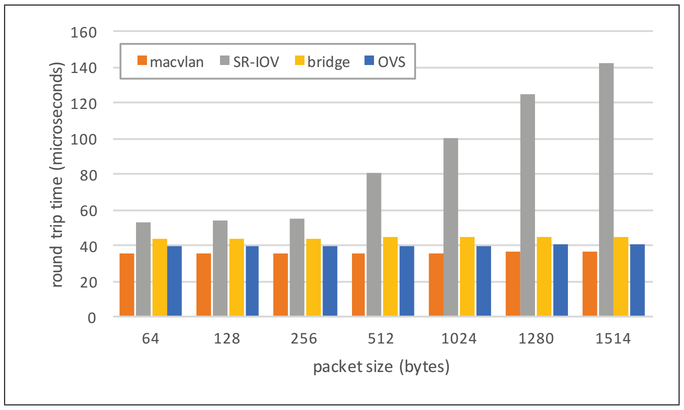

Fig. 6. Round trip time of various-sized packets between two containers on the same host, classified by networking technology.

and other kernel threads. We found that SR-IOV had nearly the same computational efficiency as direct assignment of the interface, while macvlan, the Linux bridge, and OVS increased overhead per packet substantially (11.2%, 53.4%, 26.6% respectively). As SR-IOV does not involve the CPU in packet switching, the low overhead is expected.

B. *VNF Chain Performance*

In addition to the latency cost imposed by each networking technology on packets to and from the outside network, we measured the cost of transfers between containers on the same host. This could be an important metric for optimally deploying VNF instances, as co-located containers should save physical network traffic. For this experiment, we measured the round trip time between containers using each of the different networking technologies.

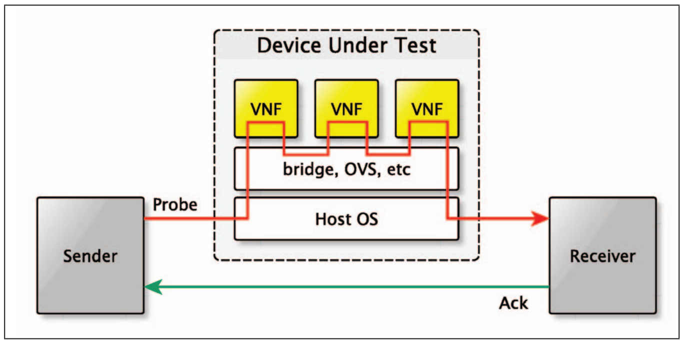

Fig. 7. Experiment architecture for simulated chains of VNFs. Probe packets emitted by the sender are routed through each VNF in the chain. Responses by the receiver bypass the VNF host.

As shown in Fig. 6, the macvlan bridge is a very efficient mechanism. It becomes apparent with these results, however, that a naïve approach can be detrimental. SR-IOV, which is very efficient at egressing packets to the physical interface, incurs a large cost to transport packets across the PCI bus for classification by the NIC. Furthermore, the cost seems to increase linearly with packet size with SR-IOV. This suggests that other, perhaps hybrid schemes for packet forwarding may be required to take advantage of SR-IOV in certain service chain deployments.

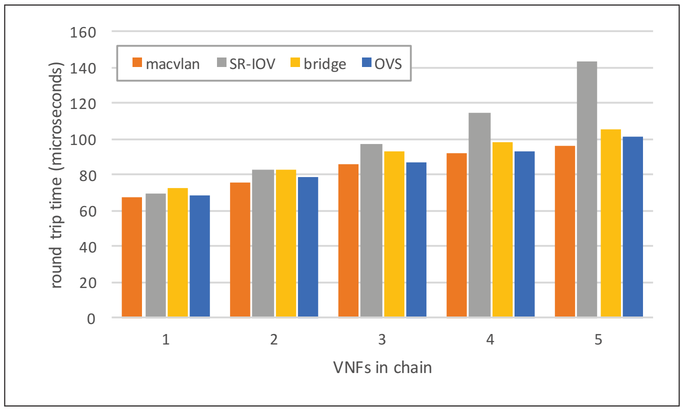

Fig. 8. Relative delay cost of packet-forwarding chains of VNFs on one host, classified by networking technology.

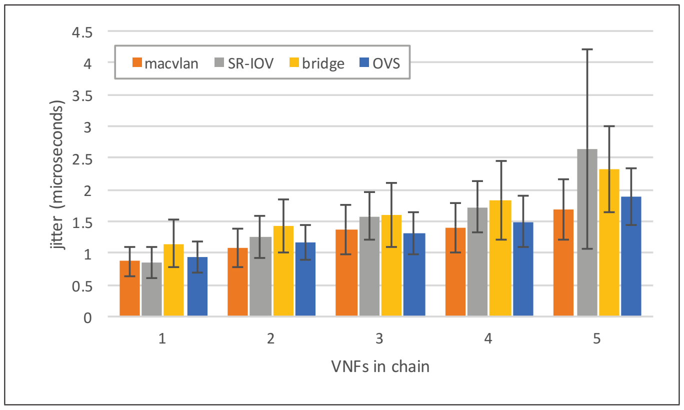

Fig. 9. Delay variation of varying-length packet-forwarding chains of VNFs on one host, classified by networking technology.

We next evaluated the networking technologies for performance in varying length chains of packet processing functions. For these experiments, we used the platform shown in Fig. 7, where packets are sent from the client, routed through a series of VNFs on the container host, and received by a server. The round trip is completed by directly sending replies from the server to the client without routing through the container host.

To prototype VNF logic, we used the Click Modular Router [22]. Click is a software package that uses a GraphML-like script language to configure and link packet processing components. This method of linking components gives it the speed of any compiled packet processing program, with the flexibility of a scripted language. We used Click to write a simple VNF that rewrote source and destination MAC addresses on each packet and forwarded them to the next VNF in the chain.

TABLE III. SERVICE CHAIN LENGTH CORRELATION (R^2)

	macvlan	SR-IOV	bridge	OVS
round trip time	0.968	0.970	0.981	0.994
packet delay variation	0.963	0.920	0.965	0.964

With each network technology, we observed a linear correlation ($r^2 \geq 0.968$) between the round trip time of packets and the number of VNFs in the service chain they passed through, detailed in Table III. We also observed linear correlation

($r^2 \geq 0.963$) between packet delay variation and the length of the chain for macvlan, bridge, and OVS. The jitter measured using SR-IOV did not correlate as well ($r^2 = 0.920$), and was more variable, likely due to the naïve intra-host implementation being tested. Our results, visualized in Figs. 8 and 9, show that the differences in delay and jitter are magnified by multiple containers running on the same host. We also noted that the Linux bridge had substantially higher delay variation in these experiments than in the interhost configuration, perhaps suggesting that the variation occurred in passing packets from the bridge to the network interface.

IV. Conclusions

Our results support our hypothesis that OS-level virtualization can offer network performance benefits compared to hardware virtualization. In every case, Docker containers have lower latency cost and lower variability than equivalent Xen VMs running the same software.

In the case of ingress and egress of packets over a physical network interface, both macvlan and SR-IOV show lower mean latency and more predictable variation than the Linux bridge and OVS. Furthermore, in both the bridge and OVS we observe latency spikes in the milliseconds that could affect packet deadlines.

Between containers on the same host, we observe that OVS produces similar delay cost to macvlan, while SR-IOV incurs much larger costs by offloading the switching to a PCIe device. This result seems to be amplified by service chaining VNFs, while macvlan and OVS perform similarly even in longer chains.

In our evaluation, we find that while the Linux bridge and Open vSwitch have acceptable performance for many applications, the demanding use case of NFV is sufficient motivation to seek more efficient technologies. We hope that this research motivates further investigations into efficient network I/O in the NFV context and other areas where lightweight containers can provide more efficient virtualization.

V. Related Work

There exists a small body of research related to high performance networking of applications in containers. [23] compares the performance of LXC containers to KVM VMs using macvlan and SR-IOV. This work focuses on namespaced routing within the Linux kernel, and has limited applicability to user-space applications typically required by more complex VNFs in containers. In a previous study [24], the same team performed an in-depth analysis of SR-IOV and macvlan for network device virtualization, but did not examine user-space networking.

Other works [25], [4], [26], [27] have compared the networking aspects of OS-level virtualization solutions, and explore the mechanisms for container network isolation that lead to a general performance advantage. However, these studies do not consider alternative I/O solutions to the Linux bridge.

Some studies have explored the performance benefits of advanced networking technologies in more traditional contexts. In [28], the authors use DPDK to allow a polling L2 switch to service many (thousands) of tiny service-chained network functions, using a variety of memory access and blocking schemes. The NFs in their work reside in user space on the host machine, which theoretically should provide similar performance to containerization. However, the need for resource isolation and security warrant an investigation of similar technology in the container context.

We believe that our work is complementary to previous work in the field, in that it sheds light on alternative mechanisms designed for use with paravirtualization that can provide benefits in use cases where high networking performance is required.

VI. Future Work

One area which deserves more consideration is the effect of latency and jitter on packet throughput and predictable delays in a VNF service chain. A related direction that could be explored is how packet train dispersion [29] is affected by factors such as chaining VNFs with multiple stages of buffering and transmission burst compression due to virtualization [18]. Connecting traffic burstiness to real world NFV use cases is an area of future inquiry.

While we took great care to use CPU pinning to make our results as reproducible as possible, we did note that the context switches, cache coherency, and CPU package colocation of associated user processes, hardware interrupt servicing, and other kernel threads can affect the latency and throughput of packet processing applications. We expect that variations in packet latency will only grow as hosts become loaded, as noted in analysis of PlanetLab [17]. There is a large body of existing work on multicore scheduling and performance, and more work is needed to connect this body of knowledge to the new packet processing and orchestration challenges posed by NFV.

One notable networking technology that may be important to the feasibility of NFV on standard x86 hardware is Intel's Data Plane Development Kit (DPDK)[11], a multicore programming framework that is most notable for its high performance poll mode driver (PMD). The PMD allows for scheduling CPU cores to the task of polling the network device and delivering packets directly to the application, without the use of kernel interrupts. In online sources and our own preliminary experiments, this method has shown to be many times more efficient in situations where the network link is highly utilized. The NFV industry has shown great interest in DPDK, and some academic exploration has been undertaken [30], [28]. The evaluation of DPDK and other similar userspace I/O technologies in relation to containers is another area of future inquiry.

Acknowledgment

This work was supported in part by funds from Dell Corporation and through NSF #1523314. This work does not necessarily represent the opinions of the National Science Foundation. We gratefully acknowledge contributions from Prof. James Martin.

[11]http://dpdk.org

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[2] ETSI, "Network Functions Virtualisation: An introduction, benefits, enablers, challenges and call for action," 2012. Available at https://portal.etsi.org/nfv/nfv_white_paper.pdf.

[3] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in Xen-based virtual cluster environment," in *High Performance Computing and Communications (HPCC), 12th IEEE International Conference on*, pp. 273–280, IEEE, 2010.

[4] S. Soltesz, H. Ptzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 275–287, ACM, 2007.

[5] J. Fink, "Docker: a Software as a Service, Operating System-Level Virtualization Framework," *Code4Lib Journal*, vol. 25, 2014.

[6] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, pp. 85–92, 2008.

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

[8] S. Yangui, M. Mohamed, S. Tata, and S. Moalla, "Scalable service containers," in *Cloud Computing Technology and Science (CloudCom), IEEE Third International Conference on*, pp. 348–356, IEEE, 2011.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[10] N. Regola and J.-C. Ducom, "Recommendations for virtualization technologies in high performance computing," in *Cloud Computing Technology and Science (CloudCom), IEEE Second International Conference on*, pp. 409–416, IEEE, 2010.

[11] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley, "Evaluating Xen for router virtualization," in *Computer Communications and Networks (ICCCN), 16th International Conference on*, pp. 1256–1261, IEEE, 2007.

[12] L. Fang, R. Zhang, and M. Taylor, "The evolution of carrier Ethernet services — requirements and deployment case studies," *Communications Magazine, IEEE*, vol. 46, no. 3, pp. 69–76, 2008.

[13] J. T. Yu, "Performance evaluation of Linux bridge," in *Telecommunications System Management Conference*, 2004.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] G. Wang and T. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *Computer Communications (INFOCOM), International Conference on*, pp. 1–9, IEEE, 2010.

[16] R. Ricci and E. Eide, "Introducing CloudLab: scientific infrastructure for advancing cloud architectures and applications," *;login: the USENIX Magazine*, vol. 39, no. 6, pp. 36–38, 2014.

[17] N. Spring, L. Peterson, A. Bavier, and V. Pai, "Using PlanetLab for network research: myths, realities, and best practices," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 17–24, 2006.

[18] J. Whiteaker, F. Schneider, and R. Teixeira, "Explaining packet delays under virtualization," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 38–44, 2011.

[19] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices," March 1999. RFC 2544.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," July 2003. RFC 3550.

[21] ETSI, "Network Functions Virtualisation (NFV): Service Quality Metrics," *ETSI GS NFV-INF 010*, 2014. Available at http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/010/01.01.01_60/gs_NFV-INF010v010101p.pdf.

[22] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[23] M. S. Rathore, M. Hidell, and P. Sjdin, "KVM vs LXC: Comparing Performance and Isolation of Hardware-assisted Virtual Routers," *American Journal of Networks and Communications*, vol. 2, no. 4, pp. 88–96, 2013.

[24] M. S. Rathore, M. Hidell, and P. Sjodin, "PC-based router virtualization with hardware support," in *Advanced Information Networking and Applications (AINA), IEEE 26th International Conference on*, pp. 573–580, IEEE, 2012.

[25] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference on*, pp. 233–240, IEEE, 2013.

[26] F. L. Camargos, "Virtualization of Linux servers," in *Proceedings of the Linux Symposium*, 2008.

[27] D. Beserra, E. D. Moreno, P. T. Endo, J. Barreto, D. Sadok, and S. Fernandes, "Performance analysis of LXC for HPC environments," in *Complex, Intelligent, and Software Intensive Systems (CISIS), Ninth International Conference on*, pp. 358–363, IEEE, 2015.

[28] I. Cerrato, M. Annarumma, and F. Risso, "Supporting Fine-Grained Network Functions through Intel DPDK," in *Software Defined Networks (EWSDN), Third European Workshop on*, pp. 1–6, IEEE, 2014.

[29] R. Jain, S. Routhier, *et al.*, "Packet trains–measurements and a new model for computer network traffic," *Selected Areas in Communications, IEEE Journal on*, vol. 4, no. 6, pp. 986–995, 1986.

[30] G. Pongracz, L. Molnar, and Z. L. Kis, "Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK," in *Software Defined Networks (EWSDN), Second European Workshop on*, pp. 62–67, IEEE, 2013.