Design Challenges for High Performance, Scalable NFV Interconnects

Guyue Liu
The George Washington University

K. K. Ramakrishnan
University of California, Riverside

Mike Schlansker
Hewlett Packard Labs

Jean Tourrilhes
Hewlett Packard Labs

Timothy Wood
The George Washington University

ABSTRACT

Software-based network functions (NFs) have seen growing interest. Increasingly complex functionality is achieved by having multiple functions chained together to support the required network-resident services. Network Function Virtualization (NFV) platforms need to scale and achieve high performance, potentially utilizing multiple hosts in a cluster. Efficient data movement is crucial, a cornerstone of kernel bypass. Moving packet data involves delivering the packet from the network interface to an NF, moving it across functions on the same host, and finally across yet another network to NFs running on other hosts in a cluster/data center. In this paper we measure the performance characteristics of different approaches for moving data at each of these levels. We also introduce a new high performance inter-host interconnect using InfiniBand. We evaluate the performance of Open vSwitch and the OpenNetVM NFV platform, considering a simple forwarding function and Snort, a popular intrusion detection system.

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; **Network performance evaluation**;

KEYWORDS

Network Function Virtualization; Service Chaining; RDMA

ACM Reference format:
Guyue Liu, K. K. Ramakrishnan, Mike Schlansker, Jean Tourrilhes, and Timothy Wood. 2017. Design Challenges for High Performance, Scalable NFV Interconnects. In *Proceedings of KBNets '17, Los Angeles, CA, USA, August 21, 2017,* 6 pages.
https://doi.org/10.1145/3098583.3098592

1 INTRODUCTION

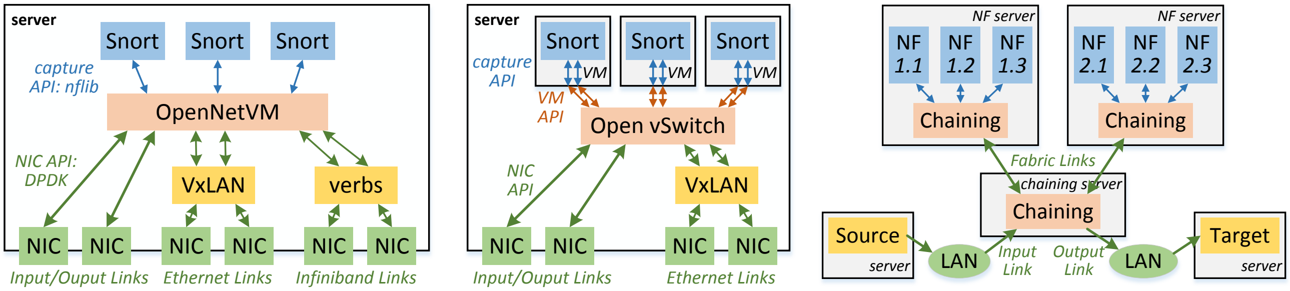
Network Function Virtualization (NFV) has enabled network services to be run on commodity servers, and replaces specialized hardware appliances. These NFs implement a range of functionality starting from security (Firewalls, Intrusion Detection Systems) to improving performance (caches) and support for new applications and protocols (TLS proxies) [10]. To support complex services a sequence of NFs may be stitched together to create a service chain. It is also important to have high performance as these services act as a 'bump-in-the-wire', requiring the components of the chain to process packets at 10Gbps and beyond. Last, but not least, scale in terms of handling a large number of flows and many different complex service chains is key to making NFV deployable in large-scale networks. Consequently, the NFV service chains may require the use of multiple servers to process the packet flow at the requisite scale and speed.

There are several possible ways to provide these large scale middlebox functions. One approach is to have the NFs running in containers or virtual machines on one or more hosts. This can accommodate existing open-source middleboxes such as the Snort [5] intrusion detection system, without rewriting NFs from scratch. To interconnect these NFs, one may use Open vSwitch (OVS) as an SDN-controlled switching fabric [4] that can forward packets through a chain. However, this approach can incur performance overheads when packets arrive at high speed, and limits the service chain flexibility since rules are typically pre-installed.

Recently, new NFV platforms [6, 8, 9, 14] have been proposed to accelerate individual NF processing as well as moving packets between NFs. Compared to a general purpose switch such as OVS, an NFV platform can be architected for more efficient and flexible service chaining. For example, NFV platforms provide shared memory-based communication [8, 14], automatic load balancing [14], and automated placement [9, 13]. These platforms can easily provide line rate performance when built upon high performance I/O libraries such as DPDK [1] or netmap [11]. However, they are typically limited to a single host and have only been evaluated using simple NFs because they require modifications to legacy middleboxes.

Supporting complex service chains and maintaining high performance requires efficient data movement. This begins with delivering data from the NIC to the NF, being able to move the packet data between different NFs, and finally moving it across hosts. In this paper we characterize the performance of packet movement at each of these levels using different technologies. The ability to load balance the workload across multiple NF instances, and implementing the service chain by having the NF pipeline span multiple hosts substantially enhances the scalability of the NFV platform. A concern however when multiple hosts are used is the additional overhead involved when going across hosts, and more importantly, the latency penalty involved. We meet the challenge of chaining across hosts by utilizing a new InfiniBand-based interconnect that supports higher throughput and lower latency, generally at lower cost per switch port than Ethernet. We provide preliminary results

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KBNets '17, August 21, 2017, Los Angeles, CA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5053-2/17/08…$15.00
https://doi.org/10.1145/3098583.3098592



(a) OpenNetVM

(b) Open vSwitch

(c) Testbed Setup

Figure 1: *(a) OpenNetVM supports shared memory-based communication between NFs that run as separate processes or containers. Either VXLAN over Ethernet or InfiniBand verbs are used for multi-host service chaining. (b) OVS supports several different VM/capture APIs that determine how packets move from the host-based switch to the NFs. VXLAN encapsulation is used for inter-host service chains. (c) Our testbed runs NFs on two NF Servers, with a Chaining Server acting as a load director between them.*

on how this interconnect can achieve the goals of scalability and high performance.

Our measurement study focuses on running real network functions in a realistic, multi-host environment. We show the importance of kernel bypass, even for heavy weight NFs, and the benefits of an optimized switching layer for building chains of network functions. We have extended the OpenNetVM platform with a unified communication interface allowing packets to be easily routed to NFs across multiple hosts over either Ethernet (with VXLAN encapsulation) or InfiniBand. Our results show the latter can improve latency and provide a notable bandwidth increase which is especially important when NFs are placed in a way that requires multiple fabric traversals to complete a service chain.

In the next section we overview the design of OpenNetVM's unified NFV interconnect and compare it to Open vSwitch. We then describe the testbed used for our measurement study before considering performance of network function service chains on a single or multiple hosts. Finally we discuss the primary insights from our results and conclude.

2 NFV INTERCONNECTS

We now describe how packets flow through NFs in our OpenNetVM NFV platform and the Open vSwitch environment, when scaling service chains across one or more hosts.

2.1 OpenNetVM

OpenNetVM provides an NFV management layer for flexible and efficient service chaining, shown in Figure 1(a). OpenNetVM supports a unified abstraction to represent both intra-host and inter-host links, providing several benefits: it grants NFs a convenient abstraction so that they can direct packets to other services with no knowledge of service placement; it hides the complexities of the underlying interconnection technologies from the NF and offloads the performance optimization tasks to the underlying NFV framework; and it allows new interconnect technologies to be used.

Our original OpenNetVM prototype [14] only supported an intra-host link using shared memory. In this work we add support for inter-host links (termed "Fabric Links") using either Ethernet or InfiniBand. In all cases, OpenNetVM leverages DPDK for kernel bypass, allowing packet data to move directly from the NIC to userspace memory pools and zero-copy transfer between NFs. NFs specify the next-hop destination NF by a service ID in the packet descriptor returned to the manager after processing, which then determines the right interconnect to use to reach an NF with the associated service ID (e.g., shared memory for an NF on the same host, Ethernet or an InfiniBand link for an NF on a remote host).

Single-Host Shared Memory: OpenNetVM leverages DPDK's support for shared pools of huge pages across processes (or containers), allowing zero-copy access to packet data. Ring buffers used between the management layer and each NF exchange lightweight packet descriptors with meta data indicating the next steps in the service chain. This can be used for either **serial chaining**, where packets go through a series of different NF services, or **load balancing**, where packets are split up and sent to one of several replicas of a service chain. OpenNetVM supports a flow table defining the service chain for each flow, and will automatically distribute flows to NF replicas of the same service based on a symmetric 5-tuple receive side scaling (RSS) hash. However with this approach, contention on shared data structures and NUMA effects can reduce performance when scaling beyond a small number of processing cores.

Multi-Host Ethernet: Ethernet is the typical interconnect used to cross hosts for scaling. However, supporting flexible service chains spanning hosts requires the outgoing packets be encapsulated (e.g., with a VXLAN or NSH header) to ensure proper routing (indicate the next-hop's host) and to indicate how packets should be processed at the next step in the chain. We have implemented a VXLAN overlay encapsulation protocol to be compatible with existing networks and route packets on any Ethernet based fabric. The OpenNetVM manager is responsible for setting up VXLAN tunnels and selecting the right tunnel to exchange packets and metadata. We have expanded the OpenNetVM manager to contact a Zookeeper based registry of NF services, allowing it to know which service types are available on each host. We use the flow hash for deciding which replica to send to, when a service exists on several hosts. More sophisticated NF selection and placement is desirable (our future work). This approach of using Ethernet is likely to incur overheads, including additional latency.

Multi-Host InfiniBand: InfiniBand is another approach to interconnect hosts in the data center and has been used in high performance computing. Compared to Ethernet, it features higher bandwidth, lower latency, and supports RDMA capabilities, while incurring low CPU overhead for messaging. These features make it possible to improve network performance [7, 12] and attractive for NFV service chaining, especially when service chains require packets to traverse multiple hosts. If the placement of NFs requires a flow to bounce back and forth between several hosts to implement a service chain, it is highly desirable that the interconnect fabric linking the NFV cluster have higher bandwidth than the input traffic link and also have low latency.

InfiniBand hosts communicate using queue pairs (QPs), and InfiniBand NICs usually support several different transport types and 'verbs'. In our current implementation, the NF manager configures QPs and uses the SEND verb with unreliable datagram (UD) to pass data (note that the VXLAN encapsulation for Ethernet is UDP-based which is also unreliable). One limitation of our current approach is that the packet pools for DPDK and InfiniBand are separate, requiring a copy to cross the boundary; we believe that this could be optimized away in our future work.

2.2 Open vSwitch

Open vSwitch (OVS) is a popular software switch that can be used in NFV deployments [10]. OVS can be used to switch traffic between VMs on a single host or between different hosts. OVS is managed by an SDN control-data plane interface such as OpenFlow [3] for installing rules. Rules can be proactively setup to support serial chaining or load balancing. We run NFs within Virtual Machines (VMs), using QEMU/KVM. As shown in Figure 1(b), OVS has several interfaces, from NIC to OVS, from OVS to VM, and from VM to the NF, all of which can impact performance.

NIC and NF APIs: Unlike OpenNetVM which is tightly integrated with DPDK, OVS supports a kernel- or DPDK-based NIC interface. We tested two flavors of OVS. **OVS-kernel**, the traditional version of OVS using a Linux kernel module, with its NIC API being the kernel driver API and its VM API being vhost-net [2]. The other OVS flavor is **OVS-DPDK**, which offers kernel bypass. For the latter, its NIC API is DPDK and its VM API is vhost-user [4]. This provides zero-copy I/O from the host NIC to the VM driver, but full zero-copy to the NF also requires an appropriate "capture API" inside the VM. We use the DPDK-enabled capture API for Snort to bypass the VM's kernel. Our measurement study explores the impact of the different combinations of these APIs.

Single-Host Load Balancing and Chaining: For single host experiments, we design simple OpenFlow rules to direct packets between the physical NIC ports and the VM interfaces. For serial chaining, we use rules to connect the adjacent ports of each NF. For load balancing, we use the OpenFlow "select group" construct to pick one of several actions (i.e., destination NFs) for a flow based on a static IP 5-tuple hash.

Multi-Host Chaining: For multiple *NF server* experiments, we use VXLAN encapsulation (without an NSH header) for routing packets from one NF to the next. The VNI header field is used to encode the virtual port of the VM on the destination *NF server*. The OVS switch encapsulates and forwards the packet based on rules defined with OpenFlow; the remote host's OVS then decapsulates the packet before transmitting it to the appropriate NF.

3 TESTBED SETUP

Our testbed was designed to test various configurations of NFV chaining. We investigate both chaining on a single *NF server*, i.e., scale up, and chaining across multiple *NF servers*, i.e., scale out. We also explore different configuration of NF chains as well as load balancing across NFs. In both cases, we replicate the same NF to form the chain and simplify analysis.

3.1 Network Functions and Configuration

In our experiments, we use both simple NFs such as 'L2fwd' and real NFs such as Snort (version 2.9.8.3). Simple NFs are easy to support and require minimal modifications, but they are stateless, not sensitive to packet payloads, and often have less computation than real NFs.

Snort processes Ethernet packets against a set of rules. We consider *Heavy Snort*, which uses the full set of community rules [5] and most of these rules perform string matching in the packet payload. We also tested a *Light Snort* with less computation, where packets are filtered against several IP 5-tuple rules (i.e. no payload matching).

Snort provides a Data Acquisition library (DAQ) which is an abstraction layer for packet I/O operations (i.e., its capture interface). We evaluate **AFPK** capture which gets Ethernet packets from the Linux network driver via a special socket, and **DPDK** capture which allows Snort to bypass the kernel and get packets directly from the virtual NIC. We also implemented a new DAQ for Snort so that it can get packets from the OpenNetVM NF manager.

We use OVS version 2.6.1 and QEMU/KVM version 2.6.2, where each VM has one CPU core, 1GB of memory, runs Debian Jessie and has two virtual NICs connected to OVS. We use DPDK 16.07 for both OVS and OpenNetVM. When using OpenNetVM, NFs like Snort run as a process. While this has somewhat less overhead than the VMs used by OVS, we believe this improvement is marginal compared to dominant effects of the switching plane used.

For throughput testing, we use `iperf` with 10 parallel TCP flows between the source and sink. For load balancing tests, we use 10 TCP flows multiplied by the number of NF configured, with a maximum of 127 flows. We use `netperf` for latency test, and unless mentioned otherwise, use the `TCP_RR` mode with 64 bytes packets.

3.2 Physical setup

Our testbed setup consists of five servers. Two NF servers are dedicated to run NF instances and one chaining server is used to route the packets in the service chain and load balance traffic as shown in Figure 1(c). Each NF server connects to the chaining server via a separate 10Gb/s Ethernet link and a separate InfiniBand link. We use two other servers as traffic source and target to emulate external Ethernet traffic, each server is connected to a separate

10Gb/s Ethernet LAN. For single host testing, they are connected directly to one of the NF servers, and no chaining server is used. For multi-host testing, they are connected to the chaining server. The detailed configurations are as follows:

NF Servers: 2 HP Z840: each has 128 GB RAM, two `Intel Xeon E5-2650-v3` (2.30GHz) processors providing 20 cores, running Debian Jessie with Linux kernel 4.6.0 [2], a `Intel X710-DA2` 10Gb/s Ethernet card and a `Mellanox ConnectX-3 Pro` 56Gb/s InfiniBand card.

Chaining Server: 1 HP Z840: 128 GB RAM, two `Intel Xeon E5-2643-v3` (3.40GHz) processors providing 12 cores, four 10Gb/s Ethernet ports and two 56Gb/s InfiniBand ports, with the same NIC type, OS and kernel as above.

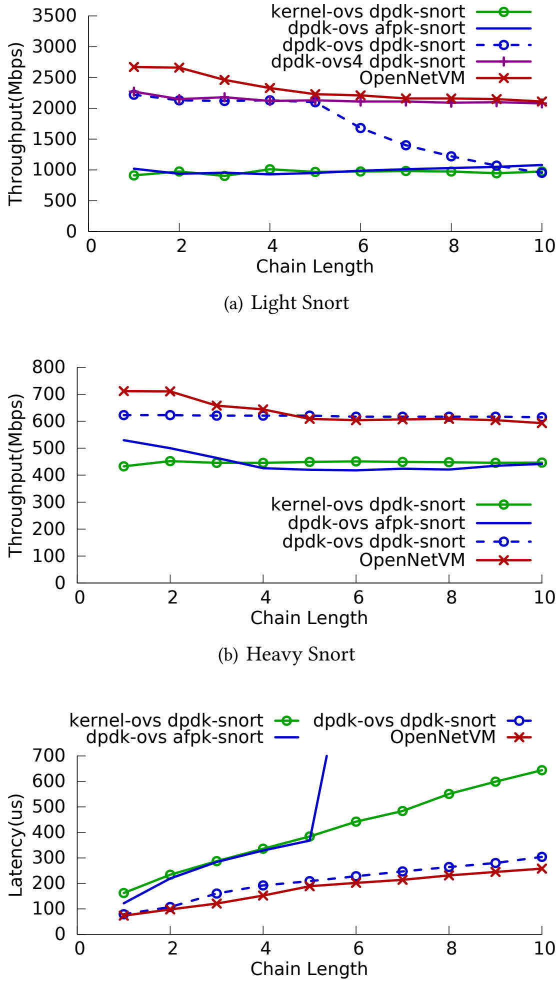
4 SINGLE HOST CHAINING

The capture and NIC APIs that an NF uses to interface with the NFV dataplane impact chaining performance. Figure 2 shows the performance of a chain of Heavy or Light Snort instances on a single server. We consider different combinations of Snort and OVS with or without kernel bypass, and our OpenNetVM platform. The *chain length* is the number of Snort instances the packets have to go through.

In Figure 2(a) we see that OpenNetVM provides higher throughput (2.1-2.7 Gbps) than full kernel bypass (*dpdk-ovs-4 dpdk-snort*, 2.1 Gbps) when running Light Snort. This illustrates the benefit of designing our architecture from the ground up for handling NF service chains, unlike OVS which is a general purpose software switch. Note that as the chain length increases, the load on the NFV switching layer rises, i.e., a chain of 10 NFs requires the dataplane to do ten times as much work per packet that enters the system compared to running a single NF. When OVS uses the default configuration (*dpdk-ovs dpdk-snort*), after five NFs, OVS becomes a bottleneck, and throughput drops linearly. Using multiple threads for OVS improves this situation; when OVS is configured to use 4 DPDK threads on 4 CPU cores (*dpdk-ovs-4 dpdk-snort*), that bottleneck is removed and throughput is flat across the chain length.

We next consider Heavy Snort, which has a maximum throughput of 712 Mbps for OpenNetVM. Due to the large amount of processing in the Heavy Snort NF, its throughput is lower than the throughput achieved by the non-kernel bypass techniques with Light Snort. Thus, we might expect that the Heavy Snort NF is the bottleneck, and data movement is a relatively smaller proportion of the overhead and adding kernel bypass in OVS should have a correspondingly small impact on throughput. But as we observe in Figure 2(b), even with the NF processing being the bottleneck, there remains a significant performance difference (as much as 40%) between the kernel and DPDK based approaches. This is shown in more detail in the table below comparing performance of a single Heavy Snort NF running on OVS with each possible combination of kernel bypass.

Using kernel bypass for either the NIC (OVS-DPDK) or the capture interface (Snort-DPDK) improves throughput and latency, while using it at both layers compounds the benefits. The fact that the benefit of DPDK is significant even with just one of the two interfaces is encouraging for multi-tenant or cloud environments where users may not have complete control over the stack—in such cases the tenant may only have control over the VM capture interfaces but not the NIC and VM interfaces, and vice versa for the provider. Naturally, full zero-copy from the NIC to the NF (i.e., Snort-DPDK and OVS-DPDK) provides the best performance, an improvement of 3.25X in throughput and 2.3X in latency compared to using kernel interfaces at both levels. OpenNetVM raises this even higher, with throughput of 712 Mbps and latency of 73μs.



(a) Light Snort

(b) Heavy Snort

(c) Latency with Heavy Snort

Figure 2: ***Longer chains increase load on the switching plane, particularly for lighter weight network functions.***

	OVS-Kernel	OVS-DPDK
Snort-NFQ	176 Mbps / 255 μs	424 Mbps / 171 μs
Snort-AFPK	192 Mbps / 179 μs	513 Mbps / 142 μs
Snort-DPDK	448 Mbps / 129 μs	624 Mbps / 76 μs

Finally, in Figure 2(c) we show the latency of Heavy Snort with each approach. Latency for *Light Snort* (not shown) is almost identical, as 64 bytes packets have nearly no payload. As expected, longer

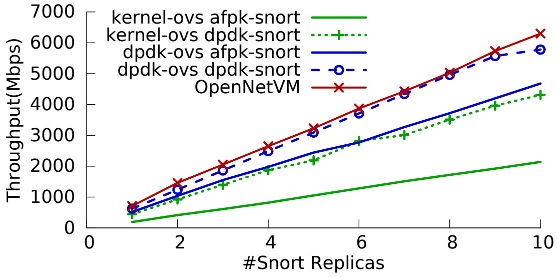


Figure 3: ***Load balancing Snort replicas on a single host.***

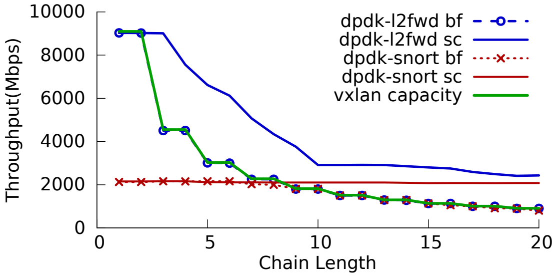


Figure 4: ***Throughput of a NF chain across multiple hosts.***

chains incur greater delay since a packet needs to go through more processing stages. The *dpdk-ovs afpk-snort* line has particularly high latency as the chain length increases. We believe this is because it adds extra load on the NF itself (which is the main bottleneck, not the OVS switching layer), leading to queue buildup in the VM's kernel.

Load Balancing To achieve line rate performance, a host might need to run multiple copies of a network function and load balance packets across them. This is shown in Figure 3 where we compare all four OVS configurations and OpenNetVM when running Heavy Snort. We see that *dpdk-ovs dpdk-snort* and OpenNetVM provide a significant improvement over the kernel based approaches–approximately a 25% increase in throughput. The figure shows a good linear speedup when increasing the number of NF replicas; however, it should be noted that this scalability relies on being able to evenly balance load across all replicas. For skewed workloads (e.g., where some flows require more processing than others), a platform such as Open vSwitch, which uses pre-configured flow rules for static load balancing, will not be able to provide good scalability since NFs could be unevenly loaded. In contrast, a dedicated NFV platform like OpenNetVM could dynamically adjust the load balancing configuration based on each NF's current load.

5 MULTI-HOST CHAINING

Having multiple NF servers allows scaling NF chains beyond what is possible with a single host. To simplify analysis, we used a single dedicated "chaining server" which sits between the two NF Servers to configure either load balancing or cross-host chains (see Figure 1(c)). However, our framework can support other configurations, such as the chaining server also hosting NFs, or multiple input and output links being distributed across the NF servers.

Chaining Placement: We implemented two policies emulating the best and worst cases for NF placement on the two NF servers with chains up to length 20. In the ideal placement, flows traverse the first 10 NFs of the chain entirely on the first server before being rerouted to the second NF server for the remaining NFs of the chain; we call this setup "Shortcut Chaining" (sc). At the other extreme, "Back-and-forth" (bf) chaining requires bouncing between NFs on the two hosts for every hop in the chain, e.g., traversing NF 1.1, 2.1, 1.2, 2.2, 1.3, etc., using the numbering of Figure 1(c). Thus, packets traverse the fabric links multiple times.

Figure 4 shows the impact of the placement and chaining policy with OVS. In this experiment, both *Light Snort* and *l2fwd* are used, and each OVS instance is configured with 6 DPDK threads on 6 CPU cores to reduce chaining bottlenecks. The back-and-forth chaining causes packets to make a round trip over a fabric link for each NF hop in the chain. The throughput of l2fwd NFs with back-and-forth policy (*dpdk-l2fwd bf*) is almost identical to the VXLAN link capacity (2*10Gbps) divided by the number of packet traversals (*vxlan capacity*), thus, the fabric link is the bottleneck. The throughput of Light Snort NFs with back-and-forth policy (*dpdk-snort-bf*) is limited by Snort processing for short chains (up to length 6, at 2.1 Gb/s), and beyond that by fabric link bandwidth. Similar experiments for Heavy Snort (not shown) show throughput mostly flat around 615 Mb/s with only a slight decrease as the chain length increases, so for Heavy Snort, there is enough fabric bandwidth and Snort's CPU processing is consistently the bottleneck. Thus, if a NFV framework needs to provide flexible NF placement and chaining/routing, sufficient fabric capacity needs to provided, *possibly well in excess of the input link rate.*

An alternative to increasing fabric capacity is smarter NF placement. The shortcut policy minimizes the use of fabric links by co-locating NFs based on their affinity. The throughput of l2fwd NFs with the shortcut policy (*dpdk-l2fwd-sc*) is higher than the back-and-forth policy, but it also reduces with chain length. When using only one NF server (chain length 1 to 10), throughput decreases quickly, indicating chaining is the bottleneck, despite OVS having 6 threads. When adding the second NF server (chain length going from 11 to 20), the first NF server is still the bottleneck. Only when the chain on the second server starts to be long is there further throughput reduction. This confirms that the amount of CPU resources dedicated to chaining needs to be balanced with the processing needs of the NFs. However, by dedicating CPU cores for chaining (i.e., OVS threads), we reduce the number of CPU cores available for NF processing.

For Light Snort with the shortcut policy (*dpdk-snort-sc*), throughput is flat around 2.1 Gb/s, which means that OVS chaining has enough resources and Snort processing itself now is the bottleneck. The performance is almost identical to the single host experiment (since we are not traversing the fabric link multiple times). Similarly, throughput for Heavy Snort (not shown) is mostly flat around 615 Mb/s, indicating that the addition of the chaining server has minimal impact on the performance for NFs having medium to heavy processing. The results so far for multi-host chaining were based on using OVS and Ethernet. We now turn to look at OpenNetVM and InfiniBand for multi-host chaining.

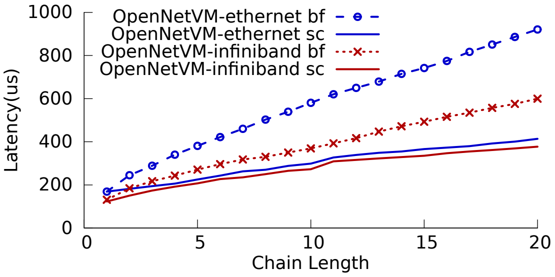


Figure 5: ***Latency of a Snort chain across multiple hosts.***

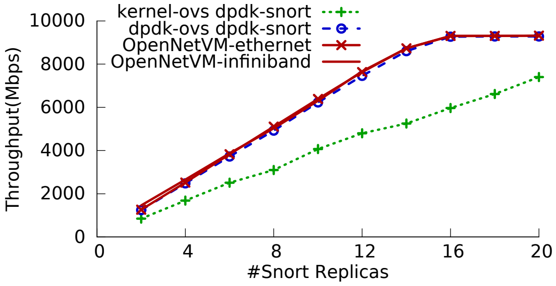


Figure 6: ***Load balancing Snort across multiple hosts.***

Fabric Type: The prior experiments illustrated that the Ethernet fabric link can become a bottleneck, particularly for back-and-forth chaining. We measure the maximum throughput with OpenNetVM's InfiniBand interconnect when directing packets through a 2 NF chain, one per NF Server. To avoid the input Ethernet link being the bottleneck we generate traffic on the chaining server itself. With 1024B packets we achieve a throughput of 14 Gbps, while 1500B packets raise the throughput to 16.7 Gbps. While our InfiniBand link supports a theoretical max bandwidth of 56 Gbps, our current implementation incurs extra copies to move packets between the InfiniBand QP and the DPDK packet data structures. Thus, such a chain implementation incurs two additional copies on each of the three servers in order to traverse the chain. We believe that this can be optimized by combining the DPDK packet pool with the InfiniBand Queue Pair data region, as well as increasing the number of QPs to increase parallelism.

Figure 5 shows the impact of the fabric type (Ethernet or InfiniBand) when running Heavy Snort chains with OpenNetVM. Latency for *Light Snort* (not shown) is almost identical, as 64 bytes packets have minimal payload. Serial shortcut chaining (sc) provides the lowest latency, since for all chain lengths, packets need to traverse the fabric network either 2 or 4 times (from the chaining server to NF Server 1 and back, and if needed to NF Server 2 and back). On the other hand, back-and-forth chaining may need up to 40 traversals for the longest chain length. Thus shortcut chaining (representing an optimal NF placement) reduces latency by between 22-35% depending on chain length. Using the InfiniBand network also provides a significant improvement, lowering latency in the back and forth case between 18-37%. These benefits compound, so using InfiniBand and shortcut chaining can provide up to a 59% benefit compared to the worst case using VXLAN over Ethernet.

Scaling Out NFs: The technique and policy for load balancing with two NF servers is the same as for a single server. We alternate hosts when adding replicas. Figure 6 shows that throughput scales up as expected up to the input line rate (10 Gb/s). With the number of instances ranging from 1 to 10, the throughput is only slightly lower than the single server case, despite adding the chaining server, the fabric links, and having a different NF placement.

6 SUMMARY

We have explored the impact of different NFV interconnect techniques with single and multiple hosts. Our measurement study provides the following insights:

- Kernel bypass provides a substantial improvement not only for minimalist network functions such as forwarding, but also for more complex network-resident functions, e.g., we get 4X the throughput with OpenNetVM compared to kernel based processing, even for heavy weight Snort which performs deep packet inspection.
- The switching layer within a host must be scalable to multiple cores to support long service chains and balance resources between chaining and processing. A general purpose software switch such as OVS can incur higher costs for NF chaining and may lack information needed for effective load balancing.
- An Infiniband-based interconnect can provide significant latency reductions and higher bandwidth, reducing the impact of NF placement decisions that require chains to be spread across multiple hosts. However, optimizing an NF platform to utilize both Ethernet and Infiniband with zero copy poses new challenges. This is our current work.

7 ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their valuable comments on this paper. This work was supported in part by Hewlett Packard Labs, NSF grants CNS-1422362, and CNS-1522546.

REFERENCES

[1] Data plane development kit (dpdk). http://www.dpdk.org/.
[2] Linux kernel. https://www.kernel.org/.
[3] Onf. openflow switch specification version 1.5.1. https://www.opennetworking.org/sdn-resources/technical-library.
[4] Open vswitch. http://www.openvswitch.org/.
[5] Snort. https://www.snort.org/.
[6] M. Honda, F. Huici, et al. mSwitch: A highly-scalable, modular software switch. In *SOSR'15*. ACM.
[7] S. Ma, J. Kim, and S. B. Moon. Exploring low-latency interconnect for scaling out software routers. In *IEEE HiPINEB*, 2016.
[8] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, et al. ClickOS and the art of network function virtualization. In *USENIX NSDI*, 2014.
[9] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A framework for NFV applications. In *SOSP'15*. ACM.
[10] A. Panda, S. Han, K. Jang, et al. Netbricks: Taking the V out of NFV. In *USENIX OSDI.*, 2016.
[11] L. Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX ATC*, 2012.
[12] T. Yu, S. A. Noghabi, S. Raindel, H. H. Liu, J. Padhye, and V. Sekar. Freeflow: High performance container networking. In *ACM HotNets*, 2016.
[13] W. Zhang, G. Liu, A. Mohammadkhan, J. Hwang, K. K. Ramakrishnan, and T. Wood. SDNFV: flexible and dynamic software defined control of an application- and flow-aware data plane. In *Middleware Conference*, 2016.
[14] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood. OpenNetVM: A platform for high performance network service chains. In *HotMiddlebox*. ACM, 2016.