Toward High-Performance and Scalable Network Functions Virtualization

Network functions virtualization (NFV) promises to bring significant flexibility and cost savings to networking. These improvements are predicated on being able to run many virtualized network elements on a server, which leads to a fundamental question on how scalable an NFV platform can be. With this in mind, the authors build an experimental platform with commonly used NFV technologies. They evaluate the NFV's performance and scalability and, based on their demonstrated improvements, discuss best practices for achieving optimum NFV performance on commodity hardware. They also reveal the limitations on NFV scalability and propose a new architecture to address them.

Chengwei Wang, Oliver Spatscheck, Vijay Gopalakrishnan, Yang Xu, and David Applegate
AT&T Labs – Research

Traditional networks consist of a large variety of dedicated elements such as routers, firewalls, and gateways. While these appliances serve well in terms of performance, managing them has been a challenge. Each device requires its own lifecycle management with dedicated staff and resources. Further, capacity provisioning has been a challenge because resources can't be shared.

By using general-purpose servers and virtualization technology, network functions virtualization (NFV) attempts to run network elements virtually on commodity platforms in a shared environment. Akin to cloud computing's impact on computing resources, NFV promises to provide significant flexibility and cost savings. In theory, it allows resources to be deployed quickly and on-demand. This vision, however, can have real-world impact only if the performance of the virtual network functions (VNFs) can match traditional network devices' performance at a lower cost. In other words, this means scaling the number of VNFs on physical servers with minimum overhead.

While the scaling and economics in conventional clouds are well understood, they don't directly transfer to NFV. NFV workloads differ from cloud workloads in the ratio of network I/O to computation resources. For instance, a typical router's data plane operation can be performed in a few cycles on a modern CPU. In contrast, processing a

Related Work in Virtualizing Network Functions

Researchers have taken multiple approaches in virtualizing network functions. Here, we discuss related work in the following categories: studying performance; hypervisors and containers; I/O virtualization technologies; software-based packet processing; and packet processing in virtual machines (VMs).

Studying Performance

Lianjie Cao and colleagues[1] investigate the performance of virtual network functions (VNFs) at the application layer, and propose a general framework for characterizing VNF performance. Our work is different in four major aspects. First, we investigate VNFs at lower layers (that is, the network layer and below). Second, our work studies configurations such as single-root input/output virtualization (SR-IOV), Data Plane Development Kit (DPDK), and non-uniform memory access (NUMA), which aren't studied in Cao's work.[1] Third, we investigate VNFs in both under – and overprovisioning scenarios, with more VNFs (63) than Cao[1] (which used up to 9 VNFs); their work only investigates the underprovisioning scenario. Fourth, we measure jitter and latency in VNFs and the throughput in our work is multiple orders of magnitude higher than Cao's.[1]

Hypervisors and Containers

Kernel-based Virtual Machine (KVM), Xen, and VMware are hypervisor solutions in virtualization. VMware and KVM can run an unmodified guest OS in VMs. Containers such as chroot Jail, FreeBSD Jail (the BSD stands for Berkeley Software Distribution), Open Virtuozzo (OpenVZ), and Solaris Container are lightweight, but they require all the VMs to run the same OS, which limits flexibility.

I/O Virtualization Technologies

Virtio emulates I/O devices in VMs, while the overheads of cross-layer translation and memory copy slow down its performance. PCI Passthrough (see http://ibm.co/1k9Uy5o) significantly improves performance by bypassing the virtualization layer – but the sharing capacity is limited because it requires assigning the whole PCI device to one VM. SR-IOV[2] and Virtual Machine Device Queues (VMDq) overcome these shortcomings with both direct access and sharing of physical I/O devices.

Software-Based Packet Processing

Click[3] is a framework to build modular routers. Routebrick[4] uses Click's program paradigm and achieves high performance and scalability by exploiting parallelism in inter- and intra-servers. Similar to DPDK, PFQ, PF RING, and Netmap[5] are frameworks for building fast packet processing. PacketShader[6] uses GPU for this purpose. Our work differs, because we focus on packet processing in NFV.

Packet Processing in VMs

ClickOS[7] builds small middlebox VMs in the Xen hypervisor. NetVM[8] builds mechanisms to improve inter-VM communications. These two works use a specialized guest OS or hypervisor while our work doesn't need modifications to the source code of OS or hypervisor. Virtual Local Ethernet (VALE),[9] Open vSwitch (see http://openvswitch.org), and HyperVSwitch[10] build software switches to create a local Ethernet among VMs. While those works focus on interVM communications, our work studies the communication performance across server boundaries.

References

1. L. Cao et al., "NFV-VITAL: A Framework for Characterizing the Performance of Virtual Network Functions," *Proc. IEEE Conf. Network Function Virtualization and Software Defined Network*, 2015, pp. 93–99.
2. PCI-SIG, *SR-IOV Table Updates ECN*, specifications library, 2016; www.pcisig.com/specifications/iov.
3. E. Kohler et al., "The Click Modular Router," *ACM Trans. Computer Systems*, vol. 18, no. 3, 2000, pp. 263–297.
4. M. Dobrescu et al., "Routebricks: Exploiting Parallelism to Scale Software Routers," *Proc. ACM Sigops 22nd Symp. Operating Systems Principles*, 2009, pp. 15–28.
5. L. Rizzo et al., "Netmap: A Novel Framework for Fast Packet I/O," *Proc. Usenix Conf. Ann. Technical Conf.*, 2012, p. 9.
6. S. Han et al., "PacketShader: A GPU-Accelerated Software Router," *Proc. ACM Sigcomm Conf.*, 2010, pp. 195–206.
7. J. Martins et al., "ClickOS and the Art of Network Function Virtualization," *Proc. 11th Usenix Conf. Networked Systems Design and Implementation*, 2014, pp. 459–473.
8. J. Hwang, K.K. Ramakrishnan, and T. Wood, "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms," *Proc. 11th Usenix Conf. Networked Systems Design and Implementation*, 2014, pp. 445–458.
9. L. Rizzo and G. Lettieri, "VALE, a Switched Ethernet for Virtual Machines," *Proc. 8th Int'l Conf. Emerging Networking Experiments and Technologies*, 2012, pp. 61–72.
10. K.K. Ram et al., "Hyper-Switch: A Scalable Software Virtual Switching Architecture," *Proc. Usenix Ann. Technical Conf.*, 2013, pp. 13–24.

request for a dynamically generated webpage will require orders of magnitude more computation cycles per network packet. The number of packets handled by a router per second is at least a few orders of magnitude higher than a Web server. This drastic shift has significant implications on NFV scalability, which hasn't been investigated comprehensively before.

To understand the implications, we evaluate the performance and scaling characteristics of

Table 1. Network functions virtualization (NFV) building blocks.

Building blocks	Options
Hardware architecture	Intel,* AMD
Virtualization	Kernel-based Virtual Machine (KVM),* VMware, Xen, Containers
Network interface controller (NIC) driver	Kernel, Data Plane Development Kit (DPDK),* Netmap,[1] PFQ
I/O virtualization	Bridge, Virtio, Passthrough,* single-root input/output virtualization (SR-IOV)[2]*

* Of the options, these are the technologies we used.

NFV. We build a test platform on commonly used building blocks in NFV, Kernel-based Virtual Machine (KVM) hypervisor, single-root input/output virtualization (SR-IOV), and Intel Data Plane Development Kit (DPDK), and subject them to various workloads across different protocol layers. We evaluate the performance in terms of not only the throughput, but also latency and jitter. To the best of our knowledge, this is the first work that investigates all three vital network metrics in an NFV performance study.

Our experiments reveal key factors and limitations in NFV performance and scalability. For instance, VNF placement is critical. Dedicating CPU resources to a VNF with respect to non-uniform memory access (NUMA) provides the best performance. However, scalability is limited, as the optimum placement isn't possible when the number of VNFs outnumbers the CPU resources. As a consequence, significant performance degradation occurs in overprovisioning scenarios. We further reason the degradation by analyzing the prolonged latency. The scheduling latency in hypervisor turns out to be a bottleneck. To address the issue, we propose an alternative NFV design that removes scheduling overhead in high-frequency packet forwarding. This method splits the routing and forwarding functions of the VNFs and moves the forwarding function to the hypervisor, which handles the forwarding tasks for VNFs. The less-frequently operated routing function is retained in each VNF. We build a prototype with results showing largely improved scalability and better performance.

This work has several contributions. The performance study provides best-practices guidance on not only how to a build an NFV platform, but also how to achieve better performance and scalability. Second, the lessons learned (along with improvement endeavors) can help NFV researchers understand the limitations and their causes, which in turn motivate more innovations to address those challenges. Finally, because our packet generator and test applications are open source, our research helps the research community to conduct similar tests easily without reinventing the wheel.

Experimental Design

To begin, let's take a closer look at the elements of our network's design.

Selecting NFV Building Blocks

Building an NFV platform involves design choices, from the hardware architecture to virtualization layers. We surveyed available options and list them in Table 1. As Intel architecture is the most widely used in industry, we picked technologies based on Intel (for instance, DPDK) so that our study will have the best generality. We chose SR-IOV with Passthrough for the I/O virtualization because it's more efficient than its peers. We won't articulate all of the technology here, due to space limitations, but for more details please see the "Related Work in Virtualizing Network Functions" sidebar, as well as respective references.

Configuring the Testbed

The testbed consists of two servers. As Figure 1 shows, one server acts as the traffic generator while the other is the *NFV server* where the DPDK-based VNF applications run in VMs or baremetal. The servers have the following hardware configurations:

- *CPUs*. There are two Intel Xeon E5-2650 2.00-gigahertz (GHz) CPUs. Each has eight physical cores – that is, 32 logical cores (lcores) with hyperthreading enabled.
- *NIC*. The network interface controller has Intel 10-Gigabit Ethernet (GbE) X520 adapters.
- *Memory*. There are eight 8-Gbyte DDR3, dual in-line memory modules (DIMMs). The total memory is 64 Gbytes.
- *Storage*. The servers use a 278.88-Gbyte SAS disk.

Servers run the Ubuntu 12.04 long-term support (LTS) operating system and are physically interconnected on one NIC port through a full-duplex 10-Gbps small form-factor pluggable (SFP+) cable. The virtualization layer uses Quick Emulator (QEMU 1.0) and Libvirt 0.9.8. Each VM has one virtual CPU (VCPU), 512-Mbyte memory, 3-Gbyte disk space, and runs Ubuntu 12.04 LTS OS. We created virtual functions (VFs) from the 10-GbE port using SR-IOV. Each VM is assigned one VF, which appears as an Ethernet interface. The maximum number of VFs on a port is 63, hence the maximum number of VMs sharing one port is 63.

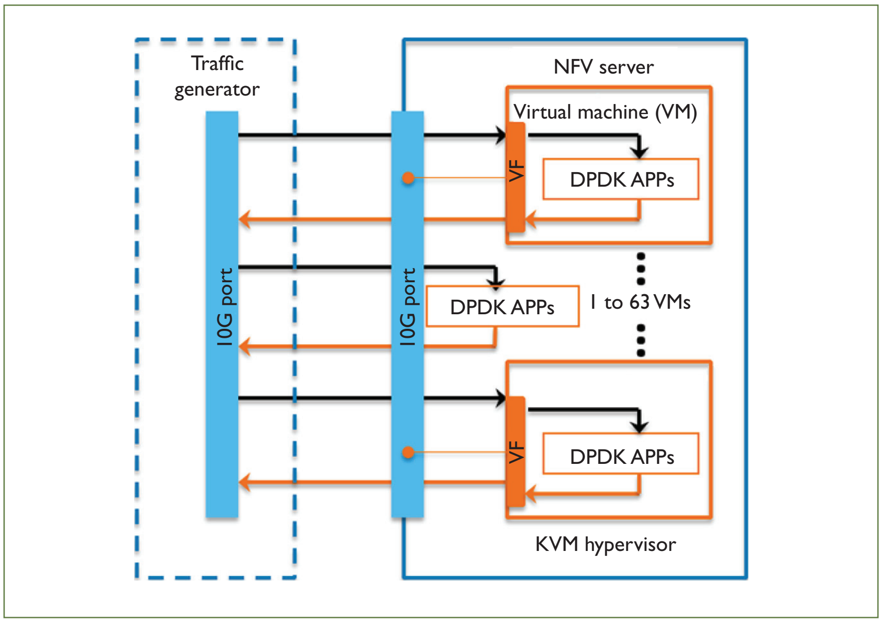

Figure 1. Testbed. One server acts as the traffic generator while the other is the NFV server. (VF stands for virtual functions.)

NFV Applications

DPDK is a set of libraries and drivers for building fast packet-processing applications. We build two NFV applications using DPDK.

Layer 2 forwarding. This reads the packet's header to get the destination media access control (MAC) address, which then is replaced with the packet generator MAC address and forwards the packet back. There's one receive (RX) queue and one transmission (TX) queue. This application has basic packet modification and forwarding operations with minimal queue/core configuration. Hence, it serves as the baseline scenario for studying software packet-processing performance in baremetal and virtualization environments.

Layer 3 forwarding using Cuckoo hashing. Compared to layer 2 forwarding, layer 3 forwarding has sophisticated packet processing and provides the router's core functionality, which basically reads a packet's IP header and searches the routing table to find the transmitting port. Example layer 3 applications in DPDK lack practicality, as they only support small routing tables. To make it more realistic and efficient, we leverage the CuckooSwitch technologies proposed by Dong Zhou and colleagues[3] and build a layer 3 forwarding application supporting 1 billion forwarding information base (FIB) entries and line-rate throughput (10 Gbps) with 256-byte or larger packets.

Layer 2 and layer 3 forwarding represent basic data plane operations shared across all VNFs: receiving, processing, and forwarding packets. Hence, our experiments reveal the VNF data plane performance on layer 3 and lower. The performance on higher layers can be studied using application-level VNFs such as video-server middleboxes,[4] which is out of the scope of this article.

Traffic Generator

Pktgen-DPDK (see https://github.com/Pktgen/Pktgen-DPDK) is a DPDK-based traffic generator running on a commodity x86 server. It generates 10-Gbps traffic with 64-byte network frames. However, it can't measure latency and jitter so we built an enhanced open source version available on github (see https://github.com/chengweiwang/Pktgen-DPDK-Latency-Jitter).

The traffic generator sends packets to the NFV server. The DPDK applications in either VMs or the baremetal server forward the packets back to the traffic generator, which then captures the performance metrics. To test layer 2 forwarding performance in baremetal, the traffic generator sends packets with a line-rate of 10 Gbps. Fixing the total traffic throughput, we vary the packet sizes from 64 to 1,024 bytes. To study VNFs, the traffic generator generates packets with the *N* MAC addresses in a round-robin, assuming there are *N* VNFs. Each VNF

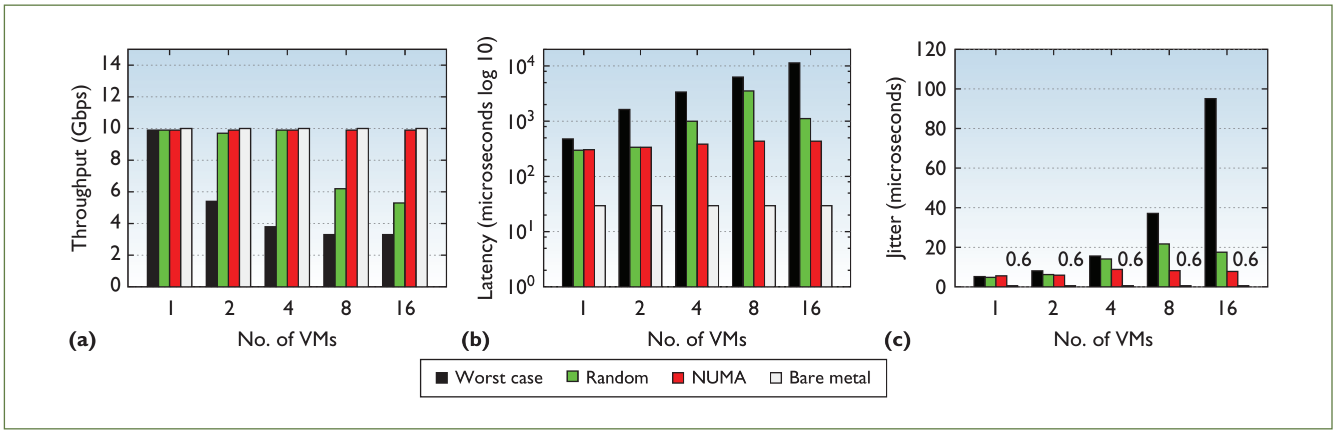

Figure 2. Layer 2 forwarding performance in the underprovisioning scenario. (a) Throughput. (b) Latency. (c) Jitter.

receives the packet with its MAC address and forwards it back to the traffic generator, which then captures the aggregated performance measurements. In the layer 3 forwarding test, besides configuring MAC addresses in a round-robin, the traffic generator randomly picks one of the *M* IP addresses in a VNF's lookup table and writes it to the packet header. In this way, lookup keys are evenly distributed among hash table entries in each VNF.

Performance Study

Here, we consider the performance of our network in a variety of scenarios.

Underprovisioning Scenario

The x86 server has a NUMA architecture, in which lcores, NIC ports, and memory units connected to the same CPU socket are considered in the same NUMA domain. Communication between different NUMA domains is through Intel QuickPath Interconnect (QPI). Accessing local memory is significantly faster than accessing memory in another NUMA domain. Our testbed has 32 lcores; 16 of them are in the same NUMA domain as the NIC port, which is shared by all the VMs. When the number of lcores is larger than the number of VNFs, there are three VNF placement strategies: the *NUMA-aware* strategy, which statically assigns a local lcore to each VM; the *random* strategy, which relies on the Linux kernel scheduler to dynamically allocate CPU resources at runtime; and the *worst-case* strategy, in which each VM was statically assigned an lcore in the remote NUMA domain.

In the layer 2 forwarding application, the packet generator evenly distributes 10 Gbps of traffic to all the VNFs. As Figure 2 shows, the VNFs have overall lower performance than baremetal, even though in the baremetal case the host uses only one lcore, while the VMs use one lcore each to forward the same amount of traffic. For instance, in Figure 2b, the latency of baremetal is lower than the latency of the VMs with any strategy. This result points to the importance of looking at all the metrics other than just throughput: if we solely looked at throughput, the performance difference between NUMA-aware and baremetal would have been negligible.

Based on the results, we learn that even with sufficient resources, NFV has lower performance than baremetal. With worst-case placement, VNFs need to access remote lcores through QPI. As VNFs compete for using QPI, the packet-processing time is prolonged, leading to queuing delay and packet loss. With random placement, the remote access penalties and QPI contention are reduced because each VNF has a probability to have a local lcore. When a VNF is allocated in a remote NUMA domain, however, the packet processing will be delayed. NUMA-aware placement dedicates one local lcore for each VNF, so it has the performance closest to baremetal. However, NFV has an extra virtualization layer, the overhead in which largely contributes to the end-to-end latency. We provide latency analysis later in the article.

From a scalability perspective, the NFV performance generally decreases as the number of VMs increases. Among the three placement strategies, scale has the smallest impact on NUMA-aware placement. In contrast, as the number of VNFs increases, the worst-case strategy's performance further declines as the resource contention on the QPI increases accordingly. The increased processing time

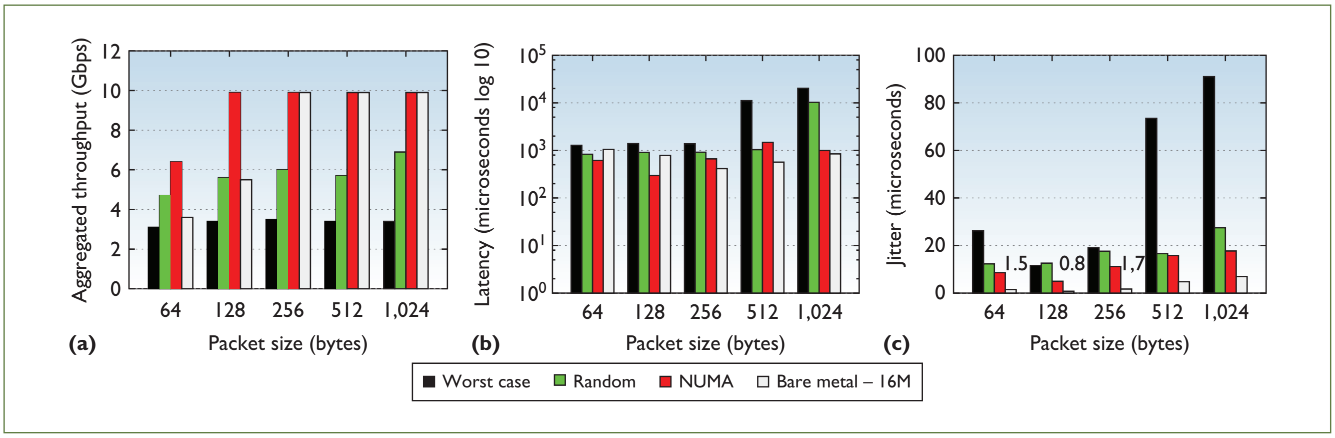

Figure 3. Layer 3 forwarding performance in the underprovisioning scenario. (a) Throughput. (b) Latency. (c) Jitter.

leads to significantly prolonged queuing delay in both hypervisor and VNF, causing about 70 percent of the packets to be dropped. From this, we learned that NUMA-aware scales well, whereas using random or worst-case strategy, packet drop, and performance degradation increase significantly as NFV scales.

In the layer 3 forwarding experiments, each VNF has 1 million entries in the lookup table and the counterpart baremetal lookup table has a table size equal to the sum of all the entries in all VNFs for each given scenario. Figures 3 illustrates the performance (as a function of packet size) of 16 VNFs and the baremetal performance with 16 million entries. The throughput increases as the packet size increases, because the number of packets decreases when the total incoming traffic is fixed at 10 Gbps. The comparison among the three allocation strategies provides the same insight we obtained on NFV performance versus baremetal. We also find that for small packets (64 and 128 bytes), 16 VNFs have better throughput performance than baremetal. The reason is 16 VNFs have 16 lcores to look up the 16 routing tables in parallel, while baremetal uses one lcore to search a 16-times-larger routing table.

We ran layer 3 experiments with different VNF quantities and found similar results that validate our lessons on NFV scalability. For brevity, we don't present the detailed results here.

Overprovisioning Scenario

In the overprovisioning scenario, where there are more VNFs than lcores, it isn't possible to pin a local lcore to every VNF as the NUMA-aware placement, with neither worst-case strategy allocating to each VM a different dedicated remote lcore. Therefore, we use the Linux scheduler here for VNF placement.

Figure 4 shows the layer 2 forwarding performance of 32 and 63 VNFs (in our test, 63 is the maximum number of virtual functions supported by SR-IOV on one NIC port). It's apparent that baremetal substantially outperforms NFV. From the layer 3 forwarding results shown in Figures 5 and 6, 32 and 63 VNFs have packet loss at all packet sizes, whereas baremetal reaches the line-rate when packet size is 512 bytes or larger. Overprovisioning also has higher latency and jitter in most cases. The 63 VNFs have slightly lower latency than baremetal with small packets sizes (64 and 128 bytes), because they use all 32 lcores but baremetal only uses one lcore. The increase in CPU resources offsets the virtualization's impact on latency.

We also study the scalability transition from underprovisioning to overprovisioning in Figure 4. We find that no more than 16 VNFs with the NUMA-aware strategy are able to sustain 10 Gpbs without packet loss. Scaling the number of VNFs over 16 can't avoid substantial packet loss, ranging from approximately 40 to 60 percent.

Hosting VNFs more than total lcores will result in NUMA violation, which delays the packet processing. The resource contention and context switch between VNFs increases the virtualization overheads. This lets us conclude that NFV performance declines substantially in the overprovisioning scenario as VNFs compete for resources and violate NUMA locality.

Latency Anatomy

To study bottlenecks causing performance degradation in overprovisioning, we first use 2.5-Gbps

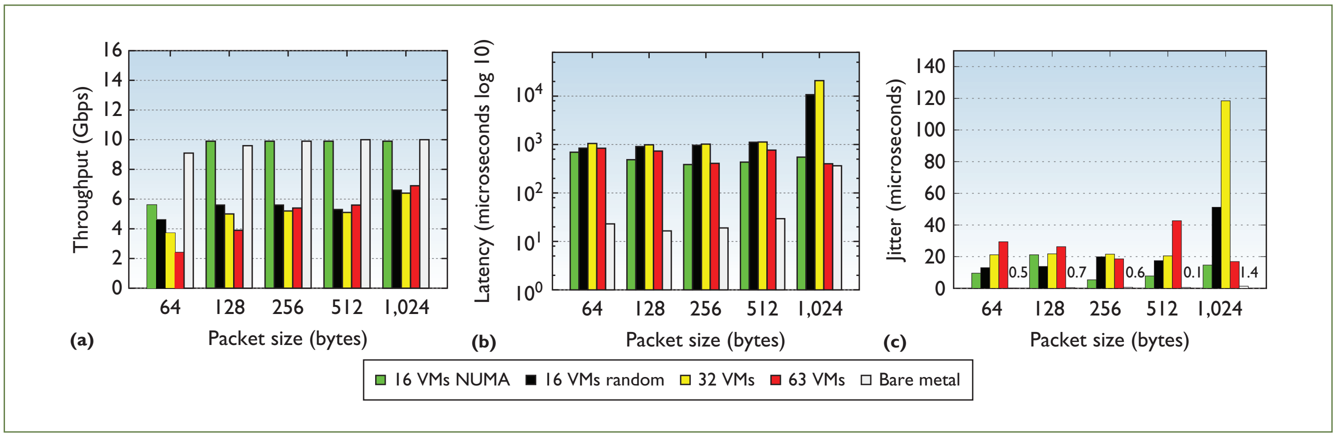

Figure 4. Layer 2 forwarding performance in oversubscription scenario. (a) Throughput. (b) Latency. (c) Jitter.

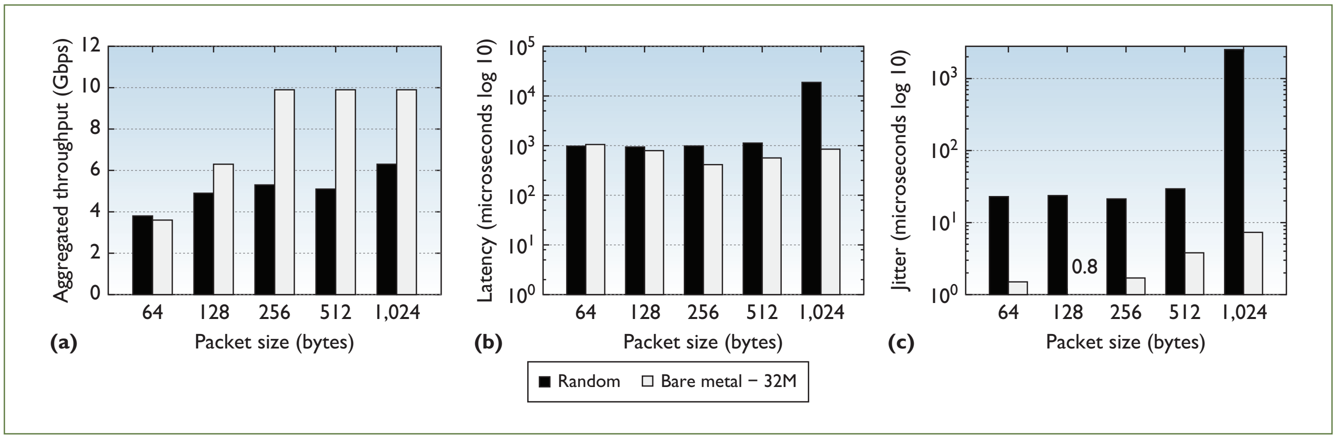

Figure 5. Layer 3 forwarding performance of 32 virtual network functions (VNFs). (a) Throughput. (b) Latency. (c) Jitter.

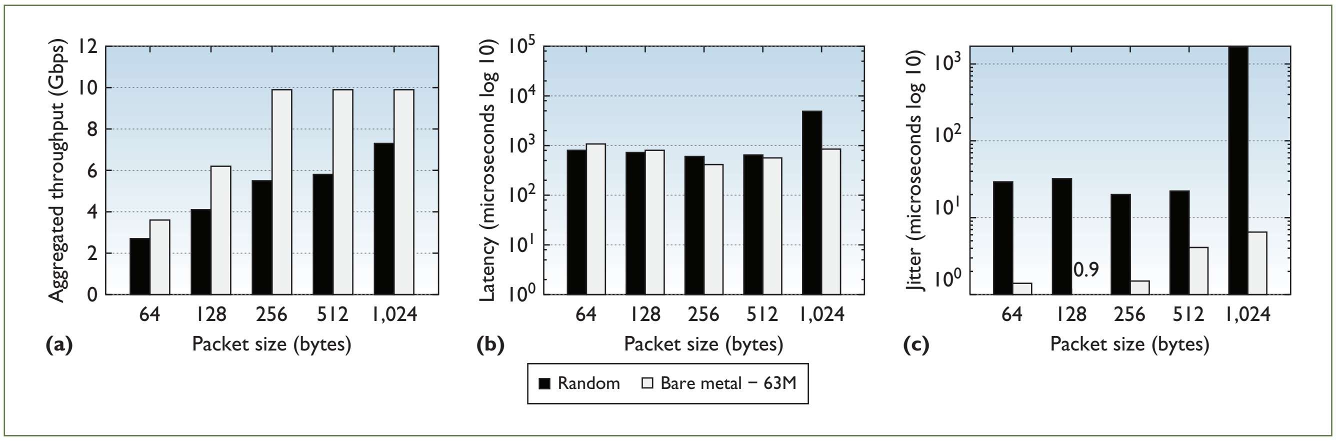

Figure 6. Layer 3 forwarding performance of 63 VNFs. (a) Throughput. (b) Latency. (c) Jitter.

input traffic (25 percent of the link capacity) to rule out traffic congestion effects. Figures 7a and 7b show the same insights learned from full load scenarios. NUMA-aware placement with dedicated cores (the underprovisioning case) outperforms other strategies as well as ones without dedicated cores (the overprovisioning case). Overall, the latency is significantly lower than the latency with full-load input traffic. Especially, the 16 VNFs with NUMA-aware strategy

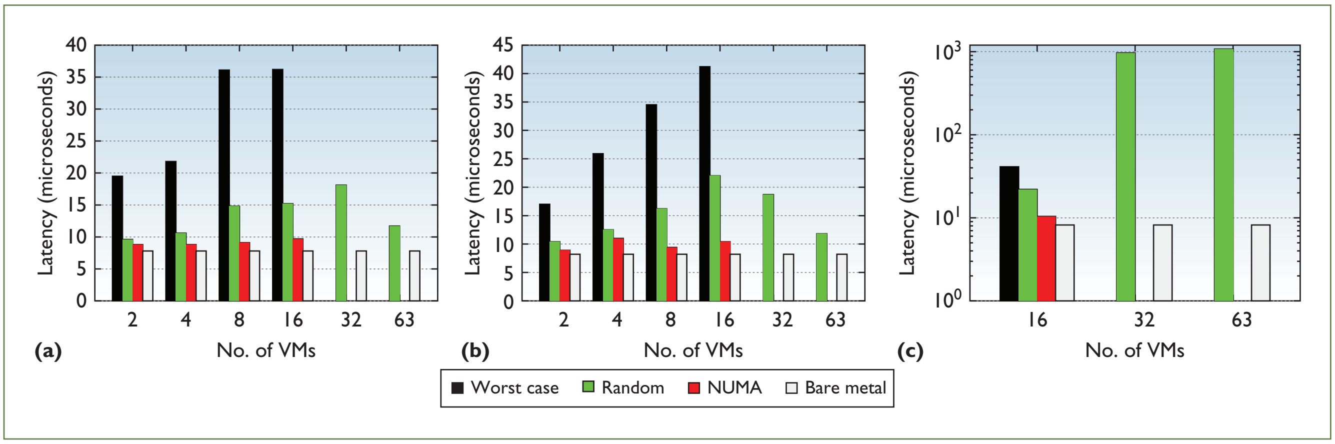

Figure 7. Latencies with various load use. Non-uniform memory access (NUMA)-aware and worst-case placements are unavailable when the number of VNFs ≥ 32. (a) Layer 2 forwarding. (b) Layer 3 forwarding. (c) Layer 3 forwarding 156 Mbps/virtual machine.

have near-baremetal latency, which suggests that NFV can perform well when traffic is low. The latency of 32 VNFs is higher than that of 63 VNFs, because their load per VNF nearly doubles. Results in Figure 7c validate the reasoning, because with the same load per VNF the latency increases as the number of VNFs increases. The same reasoning applies to Figure 2b, where the latency of 16 VNFs is lower than 8 VNFs.

The end-to-end latency includes three parts: the packet generator latency spent in the traffic generator; the VM latency spent in the VM to process the packets; and the hypvervisor latency spent in the virtualization layer for activities such as queuing and scheduling. Results in Figure 8 show that the hypervisor consumes up to 80 percent of the end-to-end latency in both the partial- and full-load scenarios. Furthermore, the percentage of hypervisor latency increases when the traffic increases from 2.5 to 10 Gbps, showing exaggerated overheads in the virtualization layer when the NFV server has more traffic.

Lessons Learned

This empirical performance study teaches us the following lessons:

- It's difficult to scale the number of VNFs to more than the number of lcores without substantial performance loss or low link use.
- A static, NUMA-aware placement strategy is crucial for achieving reliability and high performance.

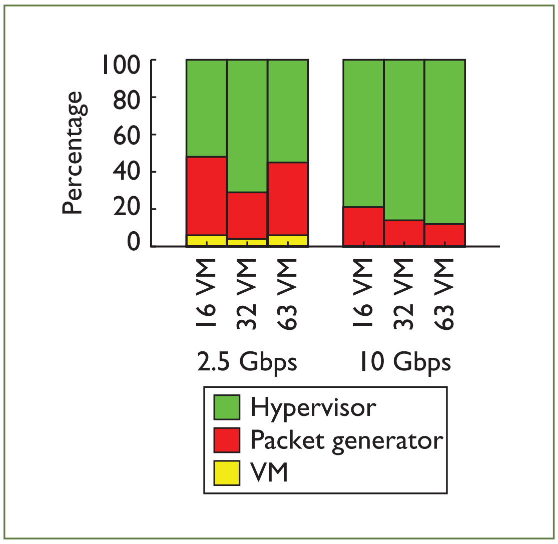

Figure 8. Latency breakdown. The hypervisor consumes up to 80 percent of the end-to-end latency in both the partial- and full-load scenarios.

Based on these lessons, we can derive a generic model to extend our empirical study to multiple NIC ports and analyze the maximum number of VNFs on any commodity NUMA server with the following: *a* ports and a maximum of *b* VFs per port; and *c* lcores on *d* sockets with *e* NIC ports per socket. We assume that each VNF only has one lcore, the server has sufficient memory and disk resources, and all packet processing is in VNF.

There are *d* NUMA domains and the number of lcores per NUMA domain is *c*/*d*. For each port

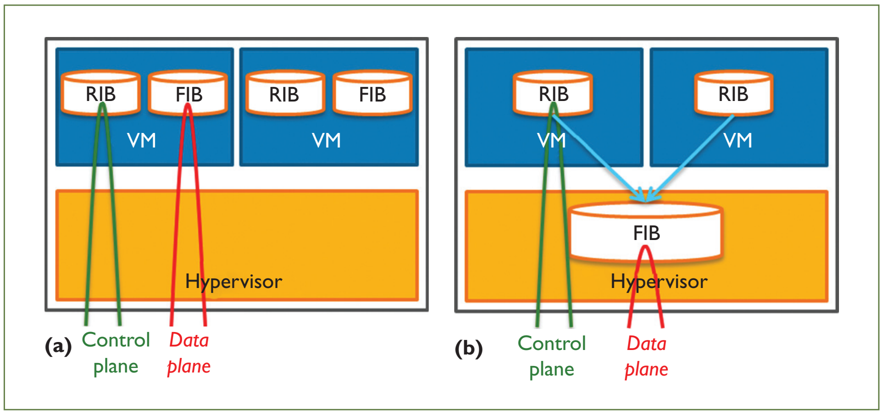

Figure 9. NFV architectures. (a) Default architecture. (b) Proposed architecture.

on a socket, the max number of lcores this port can use will be $c/d * e$. Therefore, the maximum number of VNFs sharing an individual port is $\min\{(c/d * e), b\}$, because each VNF requires one VF and one lcore to guarantee performance. Overall, the maximum number would be $a * \min\{(c/d * e), b\}$. Because $d * e = a$, the maximum number is finally $\min\{a * b, c\}$, where $a * b$ is the maximum number of VFs and c is the total number of lcores.

From this model, the number of VNFs on a server is limited by the smaller of the lcore quantity and the max number of virtual I/O functions. For example, in our NFV server, the maximum number of lcores is 32 and the maximum number of VFs on all four ports is 252 (63 $*$ 4), so we can run at most 32 VNFs on the server without substantial performance loss. Because a large server that could support 252 cores is substantially more expensive on a per-core basis and has an even more complex NUMA design than a mainstream two-socket system, the maximum number of VNFs in any practical NFV system today is limited by CPU resources rather than SR-IOV limitation. The issue will worsen because a 100-Gbps NIC entering the mainstream as the CPU scalability can't keep up with the increased rate of network speed.

Rethinking NFV Architecture

Taking the lessons learned from our study, here we consider what works best when designing an NFV architecture.

FIB Offload Architecture

A conventional NFV architecture puts the data plane function (that is, packet forwarding) and control plane function (such as routing protocols and the firewall decision engine) in VNFs. Our layer 3 forwarding application represents such an architecture (see Figure 9a). Each VNF processes volumes of packets continuously, leading to high-frequency hypervisor scheduling in an overprovisioning scenario with significant overheads (as we saw in the performance study).

To address this issue, we propose a new NFV design, aggregating VNF forwarding functions into the hypervisor while maintaining the control function within the VNFs (see Figure 9b). By doing so, the high-frequency packet processing can be handled in one place without scheduling VNFs.

This architecture is functionally similar to what Open vSwitch (see http://openvswitch.org) could support using OpenFlow[5] to separate control and data planes. However, using the long latency OpenFlow protocol will increase FIB update overheads by orders of magnitude, considering that the CuckooSwitch can support 64,000 updates per second.[3] As we show in the next section, our prototype yields better performance than Open vSwitch as well.

Prototype and Performance Evaluation

We implement a prototype using the new architecture. In the hypervisor, a layer 3 Cuckoo hashing-based forwarder processes data packets rather than relaying it to VNF. It uses the sum of all the VNF FIBs as a routing table. When control packets arrive, the forwarder sends it through a bridge to the VNF, which then writes update messages to the bridge. Upon receiving the message, the hypervisor updates the FIB.

We test the prototype's performance on 63 VNFs. The control plane traffic is set to 1 Kbps (approximately two updates per second). As Figure 10 shows, the new NFV architecture shows promising performance. The throughput and latency are close to the counterpart baremetal performance. The jitter is about 40 percent longer than baremetal, especially when the packet size is 1,024 bytes, which indicates the disturbance by update operations.

We also compare the prototype with Open vSwitch. The original Open vSwitch's (OVS) performance is significantly improved by DPDK.

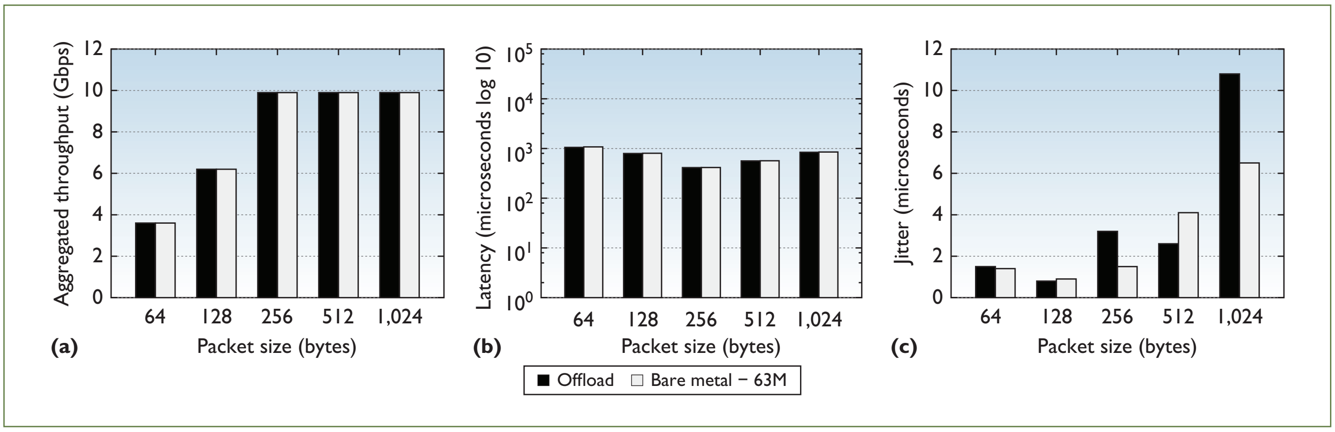

Figure 10. Offload performance. (a) Throughput. (b) Latency. (c) Jitter.

Hence, we use the DPDK-based OVS as a reference in this experiment. We test the layer 2 forwarding performance in our prototype and OVS. We configure Open vSwitch's flow table simply to forward traffic back to the traffic generator when packets are received. The results illustrated in Figure 11 shows that the offloading mechanism has generally better throughput and latency than DPDK-based OVS, especially with small-sized packets.

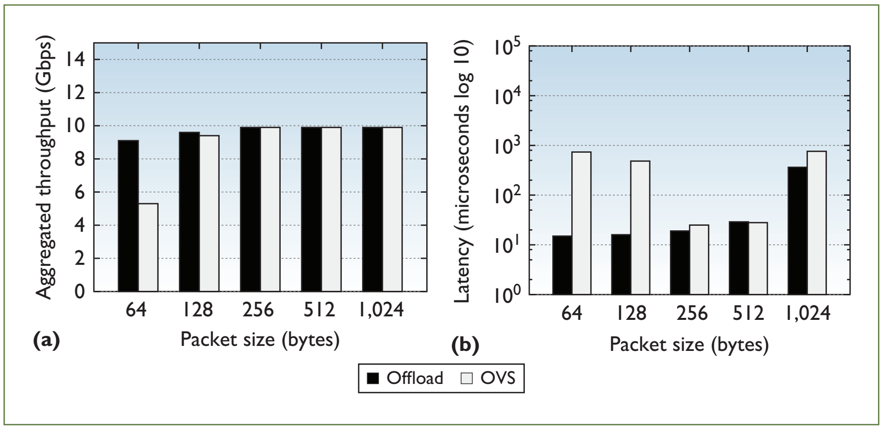

Figure 11. Offload versus DPDK-based Open vSwitch (OVS). (a) Throughput. (b) Latency.

This article studies the performance and scalability of NFV. We investigated NFV performance holistically by measuring throughput, latency, and jitter. We found that VNF placement respecting NUMA and dedicated VCPU mapping are crucial to obtain reliable and high performance. We also discerned that NFV's scaling difficulty is due to virtualization overheads. We then derived a generic model to show that scalability of NFV is essentially restrained by the number of cores on the physical server. Based on these insights, we proposed an alternative NFV architecture that offloads high volume-forwarding functionality into the hypervisor while leaving control functionality in VNFs. Our evaluation shows that this solution significantly improves performance and scalability. Going forward, we plan to further explore novel NFV architectures, aiming to build a system with full-fledged features and to compare it with other work such as Virtual Local Ethernet (VALE) and Berkeley Extensible Software Switch (BESS).

References

1. L. Rizzo et al., "Netmap: A Novel Framework for Fast Packet I/O," *Proc. Usenix Ann. Technical Conf.*, 2012, p. 9.
2. PCI-SIG, *SR-IOV Table Updates ECN*, specifications library, 2016; www.pcisig.com/specifications/iov.
3. D. Zhou et al., "Scalable, High-Performance Ethernet Forwarding with Cuckoo Switch," *Proc. 9th ACM Conf. Emerging Networking Experiments and Technologies*, 2013, pp. 97–108.
4. L. Cao et al., "NFV-VITAL: A Framework for Characterizing the Performance of Virtual Network Functions," *Proc. IEEE Conf. Network Function Virtualization and Software Defined Network*, 2015, pp. 93–99.
5. N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," *Proc. Sigcomm Computer Comm. Rev.*, vol. 38, no. 2, 2008, pp. 69–74.

Chengwei Wang is a senior inventive scientist at AT&T Labs – Research. His research interests include cloud computing, NFV, and packet processing. Wang has a PhD in computer science from the Georgia Institute of Technology.

Contact him at flinter@research.att.com or visit www.research.att.com/people/Wang_Chengwei/index.html.

Oliver Spatscheck is a research fellow at AT&T Labs – Research. His research interests include network-centric systems, network measurements, and network security. Spatscheck has a PhD in computer science from the University of Arizona. Contact him at spatsch@research.att.com or visit www.spatscheck.com/oliver.

Vijay Gopalakrishnan is a director in the Network and Service Quality Management Center at AT&T Labs – Research. His research focuses on systems challenges in the architecture, protocols, and management of networks. Gopalakrishnan has a PhD in computer science from the University of Maryland, College Park. Contact him at gvijay@research.att.com.

Yang Xu is a senior inventive scientist in AT&T Labs – Research. His research interests include software-defined networking, NFV, network protocols, and network optimizations. Xu has a PhD in operations management with a focus on network optimizations from Rutgers University. Contact him at yxu@research.att.com.

David Applegate worked on this research while he was AT&T Labs – Research, and the copyright/IP for this work is owned by AT&T. Currently, he is a researcher at Google Research. His research interests include integer programming, combinatorial optimization, discrete algorithms, and parallel optimization. Applegate has a PhD in computer science from Carnegie Mellon University. He is a member of the IEEE Communications Society and a recipient of the following prizes: Frederick W. Lanchester, William R. Bennett, and Beale-Orchard-Hays. Contact him at dapplegate@google.com.