Performance Management Challenges for Virtual Network Functions

Wei Zhang[†] Jinho Hwang[*] Shriram Rajagopalan[*] K.K. Ramakrishnan[‡] Timothy Wood[†]
[†]George Washington University [*]IBM T. J. Watson Research Center [‡]University of California, Riverside

Abstract

Networks increasingly incorporate complex functionality, going beyond simple forwarding. Packet flows require processing through a complex set of services, that may often be provided in the cloud or on commercial off the shelf (COTS) hardware. This paper investigates the problem of scheduling network functions in different service chains, on a single physical host. Stock operating system (OS) schedulers are ill-equipped to handle the strict performance required—high throughput AND low latency across multiple functions—by network function virtualization (NFV) platforms. Our experimental results show that the standard Linux scheduler can cause a 50% drop in throughput when subjected to diverse NFV workloads. We argue that the OS scheduler needs to be enhanced to be NFV-aware in order to handle service chains, while maintaining line-rate performance. We describe the challenges in designing a network function service chain friendly scheduler and present some early results based on our ongoing research in this area.

I. Introduction

Middleboxes have become a critical part of modern enterprise and data center networks, being used to improve security and performance. A 2011 study of network operators found that the number of middleboxes was on par with the number of layer-3 routers, illustrating the growing complexity of network data planes [9]. While Software Defined Networking (SDN) has provided new ways to manage the network control plane, it does not directly assist with managing the operational issues and performance of data plane middleboxes.

Because of the significant growth of middleboxes both in data centers and service provider networks, there is a need to improve how middlebox resources are managed to ensure that their performance is adequate. Concurrently, Network Function Virtualization (NFV) is a promising approach to run software middleboxes on commodity hardware, simplifying and lowering the cost of deployment. The ability to dynamically place the software-based functions as needed provides significant flexibility and enables scalability, while also reducing the need for complex network routing and maintaining low latency. For all its benefits, supporting NFV on COTS hardware does pose challenges: appropriately placing middleboxes and efficiently utilizing server resources while guaranteeing their performance needs becomes a major difficulty.

978-1-4673-9486-4/16/$31.00 © 2016 IEEE

Network functions (NFs) are sensitive to both latency and packet drops since they are typically interposed between communicating hosts. While current NFV platforms are capable of servicing rates of 10Gbps or higher with very low latency [8, 3], the performance requirement is quite stringent: each 64 byte packet must be serviced within 67.2 *ns*. For computationally-intensive NFs, this can be a challenge using a single core. Many complex NFs must be assigned multiple cores so that incoming packets can be processed in parallel. On the other hand, some NFs only need to work on a small portion of flows, so they may need substantially less than the resources of a single core.

When provisioning multiple NFs on a single physical host, a typical approach is to simply overprovision each NF to ensure it can meet its performance requirements. In NFV platforms based on the popular Intel Data Plane Development Kit (DPDK), high performance is achieved by dedicating cores to each NF and having them poll for packets [4, 3]. Other approaches such as netmap use interrupts instead of polling to reduce their energy consumption, but the recommended approach is still to dedicate cores to NFs [8]. In both cases, resources are wasted when NFs see variable workloads and do not need entire CPU cores. Dedicating cores also places a severe limit on the total number of NFs that can be run per server, making such systems very inefficient for deploying a wide variety of NFs that each only need to process a small number of flows.

The situation is further complicated by the growing complexity of NF service chains. Network carriers and data center operators combine multiple NFs in complex topologies that chain several NFs to build an overall network service [11, 7, 12, 10]. While initially service chains were relatively simple, static combinations of NFs, they have grown in complexity. Dynamic service chains pose new challenges for resource and performance management since it becomes more difficult to predict what resources will be needed in advance. Workload fluctuations or events such as a Denial of Service attack might activate a different service chain path, causing substantial variability in NF resource requirements.

Managing the resource allocation and performance of systems in this complex network context is a unique and interesting problem compared to supporting general purpose tasks on a host or in data centers. Dependencies between heterogeneous NFs in a given service chain for a packet flow imply that a bottleneck in the chain can result in significant impact in terms of packet loss, lost throughput, inefficiency and undesirable user-

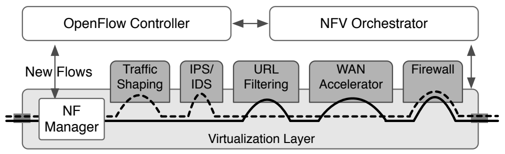

Fig. 1: NFV platform with service chains

perceived latency. Needing to maintain low latency throughout the service chain, while continuing to process packets at line rate means throughput AND latency both become important. The slack available for the operating system to do resource management and scheduling NF processing can be quite small.

In this paper we explore the factors that affect NF performance, and describe some of the resource management challenges they face. We measure and analyze a variety of interesting NF performance issues including:

- Poor CPU scheduling in Linux can lower throughput of NF chains by nearly 50% when loads are unbalanced.
- Inefficient NF placements that cause service chains to cross sockets can triple latency and reduce throughput by 60%.
- Depending on the computation costs of NFs, it can be more efficient to have a service chain share a CPU core rather than spread them across multiple cores, despite fewer resources being available.

II. NFV Platform Background

NFV platforms provide a runtime on which different NFs can be run simultaneously on the same host. The NFV platform manages all packet I/O and routes packets across different NFs present in the host according to pre-defined policies. Figure 1 shows an example platform architecture where an NF manager initially receives packets and then directs them to several NFs based on rules provided by an OpenFlow-based SDN Controller. An NFV Orchestrator instantiates new NFs across hosts as needed.

Our architecture is inspired by our prior work on NetVM [3], where network functions are packaged inside dedicated virtual machines (VM). In this paper, we run network functions as separate processes on the host OS, to facilitate studying their performance characteristics without the overhead caused by virtualization. When packets arrive at the NIC, the NF Manager uses DPDK's zero-copy I/O to DMA packet data into a memory region shared with all NF processes. The manager then notifies the appropriate NF by copying a lightweight packet descriptor into a ring buffer shared with the NF. The NF reads the descriptor in the ring to determine the packet's address in the shared memory region. After an NF processes the packet, it adds the descriptor to a second ring buffer shared with the NF Manager. The manager has a separate thread which checks this ring buffer and then either gives the packet to a different NF or sends it out the NIC depending on the flow table rules.

Polling-based NFV architectures such as those based on DPDK achieve high throughput at the expense of high resource utilization [4, 3, 7]. Interrupt-driven approaches such as netmap also can achieve line-rate packet processing, while using resources proportional to the incoming packet rate [8].

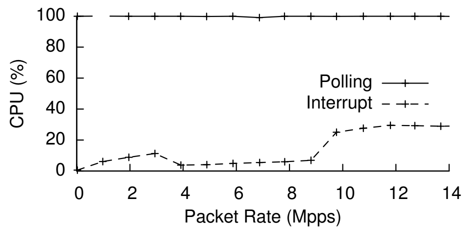

Fig. 2: CPU consumption when using interrupt and polling based NFs

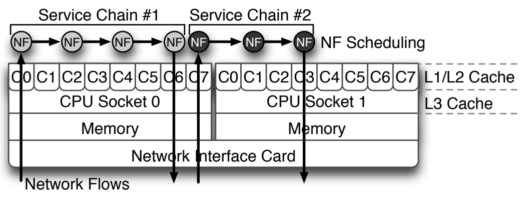

Fig. 3: NF placement on a multi-socket host

However, interrupt-driven approaches typically require large batch sizes, which can potentially hurt latency.

In our platform we consider a hybrid approach that combines polling and interrupt-driven processing. Polling is used by the NF Manager to retrieve packets from the NICs and to distribute them among NFs which can be either polling or interrupt-driven. This allows the system to easily scale to cases where the total incoming packet rate is very high, but a large number of NFs must be run, each potentially only serving a small fraction of the total rate.

Interrupt mode can be used by our NFs if it is desirable to reduce power consumption or to allow sharing of CPU resources. The interrupt driven mode uses batching rather than per-packet interrupts; an NF is only woken up once the number of packets in its arrival queue exceeds a threshold (128 packets in these experiments, but we expect to examine the appropriateness of the value of this threshold in the future). Similar to netmap [8], once an NF is active the NF Manager will not send any further interrupts, reducing communication overheads. Once an NF completely empties its queue, it will go back to sleep and wait for another interrupt.

Here we evaluate the resource utilization and throughput of both polling and interrupt based NFs. We use NFs with no computation cost, i.e., they receive a packet and then immediately send it back to the NF Manager. We measure the CPU utilization while adjusting the incoming packet rate.

A polling-based NF achieves the full line rate throughput of 10Gbps with 64 byte packets, however it also uses 100% of the CPU, regardless of the incoming packet rate. In contrast, the interrupt-driven NF is only able to achieve rates of up to 8 Gbps, but it uses the CPU in proportion to the incoming load. We expect that with further optimizations, our interrupt-driven mode will be able to achieve the full 10Gbps throughput; its relatively low CPU consumption even under full load suggests that it is unnecessarily sleeping when it should stay active to process soon-to-arrive packets.

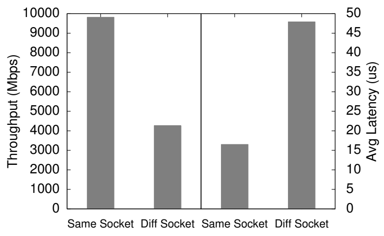

Fig. 4: Throughput (left) and Latency (right) Impact of NUMA-aware (same socket) vs. NUMA-unaware (diff socket) Scheduling.

III. NF Placement

One of the benefits of NFV is the ease with which related NFs can be deployed onto the same host, reducing network bandwidth usage and latency. A placement engine is used to decide which data centers, servers, and cores different NFs should be located on. When NFs share CPU cores, the placement problem becomes even more challenging since the resource requirements of NFs must be considered, as well as potential resource/network contention between them.

NUMA Overheads: Modern data center servers typically have multiple sockets, each hosting CPUs with multiple cores and independent memory. Since motherboard layouts place memory slots adjacent to each CPU socket, Non-Uniform Memory Access (NUMA) effects can cause overheads on latency sensitive applications [2].

Figure 3 illustrates how two service chains might be deployed on a server with two CPU sockets, each with eight cores. Here service chain 1 runs entirely on the first socket, allowing it to access all local memory, while service chain 2 is split between the two sockets, potentially adding overhead due to increased latency accessing remote memory.

Our experiments prove that NUMA overhead can have a substantial impact on NF performance, as shown in Figure 4. For this experiment, we run a single-threaded NF, and place it on either the same or a different socket than the NF Manager. The figure shows that a simple forwarding NF running in the same socket with the NF manager can achieve up to 10Gbps rate with 16.53 *us* latency, but having the NF in the other socket only achieves 4.2Gbps and increases the latency to 47.91 *us*. The obvious reason for the performance difference is the cache hit rate, which is 96% for the same socket case, and 17% for the different socket case since the NF cannot find packet data in the local socket's cache and must perform slower remote memory accesses.

This illustrates the importance of an NF placement engine being aware of the underlying hardware characteristics, as well as the service chain topology.

Service Chain Placement: A service provider network may contain many different network functions ranging from forwarding entities like switches and routers to firewalls, proxies, caches, and policy engines. Flows have to be dynamically routed through these network functions to form a service chain, which could be dictated by a software-defined network [6] or network services headers (NSH). How NFs are placed, i.e., on the same host, on the same socket, or even on the same core, is a difficult problem that can have a large impact on service chain performance.

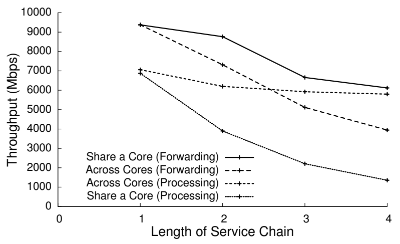

Fig. 5: Service Chain Placement: "Forwarding" does not have any computation cost, but "Processing" runs a network application that consumes 132 *ns* (354 CPU cycles) per packet.

Figure 5 illustrates the performance impact of simple service chain placement scenarios, where all NFs share one core (Share a Core) or each NF gets assigned to different cores (Across Cores). The number of NFs (length of service chain) increases up to 4, which means a maximum of 4 cores are used, or at the other extreme, at most 4 NFs share a core. Surprisingly, when NFs simply forward packets without any computation cost, sharing a core (and thus its cache) achieves better throughput than pinning NFs to different cores, despite frequent context switches in the shared case. However when NFs run network applications that require processing, the power in the shared core case becomes insufficient, so packets are dropped in the service chain, resulting in lower throughput. Assigning NFs to separate cores helps a lot to support the required processing.

Our results suggest that knowing the type of computation performed by different NFs is very important in order to maximize the hardware resource usage and in turn, performance.

IV. NF Scheduling

One of the key features of server virtualization has been the ability to multiplex a single host, and even a single core, for multiple virtual machines while still offering strong isolation. This fine-grained resource sharing is what enabled cloud computing. Network Function Virtualization can similarly benefit from resource sharing, but many existing systems explicitly avoid that in order to maximize performance. Here we study the performance impact of sharing CPU cores, as well as the limitations of current schedulers when running NFs with diverse requirements.

Previous studies of interrupt-driven NFV frameworks have focused on maximizing performance, and thus they do not consider the case where several NFs share a CPU. We have found that the standard linux scheduler, completely fair scheduler (CFS), can perform quite poorly in this scenario since it focuses on fairness.

To evaluate how computation cost affects performance, we consider a service chain of three NFs all sharing one CPU core. The first and last NF in the chain have a computation cost of about 77 nanoseconds. We vary the computation cost of the middle NF and measure the maximum throughput before packets start to be dropped.

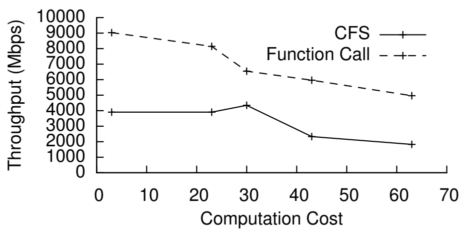

Fig. 6: Unbalanced computation costs hurt performance.

In Figure 6, the "Function Call" line indicates an idealized NF scheduler where the three NFs are actually just separate function calls made within a single process, thus there are no context switch overheads and maximum throughput is achieved since the optimal function is always called next. As expected, for this case the throughput decreases linearly as the computation cost of the middle NF rises. In contrast, when using separate NFs and the CFS scheduler, the best performance occurs when the computation cost of all three NFs are equal (30). This surprising result indicates that even if the second NF performs negligible computation before passing on a packet, it still hurts performance. This happens because CFS allocates a similar amount of CPU to each of the NFs, even if the second NF doesn't really need it. This causes the second NF to do wasted work—it processes packets faster than NF 3 can handle them, causing its queue to overflow.

To explore this further we consider a second scenario where three NFs are run on a shared core and we adjust the relative computation costs across all three, keeping the total computation constant. For this scenario, if computation is evenly divided between the NFs the performance is 4Gbps, and the Function Call based NF achieves 4.9Gbps regardless of how computation cost is split. However, if the computation cost is unevenly divided between the three NFs, the throughput drops to less than 2.1Gbps a 48% reduction.

These performance problems occur because the linux scheduler tries to give equal shares of the processor to CPU-bound processes, while giving precedence to processes responding to interrupts. Both of these features can have a negative impact on NFs. First, CPU share within a service chain should be divided up based on the computation cost (or priority) of NFs since the goal of an NF scheduler should be to give fairness on the drop rate at each NF in the chain. Second, the priority boost given to processes receiving interrupts tends to cause the first NF in a chain to get a higher share of the CPU since packets are continually arriving to it from the NIC. In contrast, other NFs in the chain will only receive packets once their predecessor in the chain is scheduled. This leads to the first NF being able to frequently steal the CPU from other NFs, which can easily lead to an overflow of the incoming queue at the second NF in the chain. This latter problem makes it so that even if priorities proportional to their computation costs are given to NFs using the `nice` command, performance still does not improve.

V. Discussion

Global placement is not enough. To achieve the best performance, NFV platforms need to be able to solve two problems: 1) where to place the NFs in a data center—a global optimization problem, and 2) how to efficiently schedule the NFs running inside a single host. We find that prior work has focused on solving the former problem but not the latter. Inefficient scheduling can lead to poor throughput/latency on a single host, that in turn can negate the positive effect achieved through efficient placement of NFs across the data center.

OS needs to be NFV-aware. For effective scheduling of NFs in a single host, NFV platforms need to be aware of both the underlying hardware (number of sockets, NUMA domains, etc.) and the service chain topology. The scheduling problems will become harder with long dynamic chains. The OS needs an NFV-aware scheduler, that is cognizant of the SLAs of each NF. We believe that exposing certain NF state, such as an NF's queue length, chain length, etc., to the kernel scheduler would enable it to schedule NFs in a way that improves throughput and latency, while maximizing system utilization.

Performance isolation with service chains. NFV platforms are multi-tenant by nature, as they run NFs from different vendors on shared hardware. Each vendor has a chain of NFs whose computational costs could vary dramatically depending on network traffic, time of day, etc. Hence, performance isolation is paramount on over-provisioned hardware to ensure NFs (and hence NF service chains) from different vendors get their fair share of system resources when needed. We are currently investigating three potential solutions for this problem: lightweight VMs [5], Docker container-based isolation, and library OS architectures [1].

Acknowledgments: We thank Chenghu He for his contributions to the early stages. This work was supported in part by NSF grants CNS-1422362 and CNS-1522546.

References

[1] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *Proc. of OSDI*, 2014.
[2] L. Bergstrom. Measuring NUMA effects with the STREAM benchmark. *CoRR*, abs/1103.3225, 2011.
[3] J. Hwang, K. K. Ramakrishnan, and T. Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *NSDI*, 2014.
[4] Intel-Corporation. Intel Data Plane Development Kit: Getting Started Guide, 2015. ONLINE.
[5] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the Art of Network Function Virtualization. In *NSDI*, 2014.
[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun.*, 2008.
[7] 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*, 2015.
[8] L. Rizzo. netmap: A novel framework for fast packet I/O. In *Proc. of USENIX Annual Technical Conference*, 2012.
[9] J. Sherry and S. Ratnasamy. A Survey of Enterprise Middlebox Deployments. Technical Report UCB/EECS-2012-24, Feb 2012.
[10] P. Veitch, M. J. McGrath, and V. Bayon. An instrumentation and analytics framework for optimal and robust nfv deployment. *Communications Magazine, IEEE*, 53(2):126–133, 2015.
[11] T. Wood, K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang. Toward a software-based network: Integrating software defined networking and network function virtualization. *Network, IEEE*, 29(3):36–41, May 2015.
[12] L. T. X. P. Yang Li and B. T. Loo. Network functions virtualization with soft real-time guarantees. In *INFOCOM, IEEE*, 2016.