openNetVM: Bringing Elasticity to Enterprise Networks using Network Function Virtualization on Commodity Hardware

Neel Shah Philip Lopreiato Warren Smith Wei Zhang Timothy Wood
The George Washington University
{nshah95, plopreiato, warrens, zhangwei1984, timwood}@gwu.edu

I. Abstract

Scalable high performance networks are essential components of enterprise and data center networks. Traditionally, high performance networks are comprised of middle-boxes which control the processing of traffic. These components tend to be expensive and inflexible. To solve the economic and scalability problems that hardware networks face, industry trends show an increase in the adoption of network function virtualization (NFV). However, these technologies do not provide the same high performance and security that hardware does.

We present openNetVM, a scalable and high performance NFV platform that provides tight control of network flow. Our platform is built with Intel DPDK and Docker containers, which allow us to achieve enterprise and data center level performance on commodity hardware. openNetVM provides utilities to create and manage NFs throughout their lifecycle.

II. Introduction

Modern enterprise and data center networks are comprised of various middle-boxes (NFs) to improve their performance, security, and quality [1]. These middle-boxes are services such as firewalls, load balancers, or monitoring utilities. Traditionally, these middle-boxes are hardware appliances, which make them expensive and difficult to upgrade [2]. NFV aims to change this by virtualizing entire groups of NFs on commodity hardware. This eliminates hardware based NFs, significantly lowering costs and increasing elasticity of the network topology [3].

For example, an enterprise network that consists of an F5 firewall, Cisco Nexus 9000, and three Dell PowerEdge R920 servers, would cost around $550,000. A network backed by NFV would only need two Dell PowerEdge R320 and two Intel x520 network cards, which costs $2,500 [6], [7], [8], [9].

Traditional NFV platforms involve creating entire virtual machines for a single application. For example, VMware vSphere enables NFV by making new virtual machines to run middle-boxes [4]. This model does not scale effectively in enterprise or data center level networks. Although a virtual machine can run middle-boxes, this approach causes overhead because virtual machine resources are not fully utilized. To avoid this, a different technology, Linux Containers, provides a platform to run processes in individual segregated environments. Docker is an open source implementations of Linux Containers.

Our open source NFV platform, openNetVM, establishes an environment to run various NFs within Docker containers. This platform is built using Intel DPDK libraries instead of the standard networking libraries. DPDK enables fast packet processing, implements a framework for multicore systems, and establishes storage mechanisms that are faster than the standard networking libraries.

As shown in Figure 1, openNetVM uses zero-copy packet forwarding, NUMA-aware processors, lockless data structures, and polling to provide high networking performance. The NF manager creates an environment for each NF by allocating memory for packets and metadata. Once the environment has been established, an NF runs

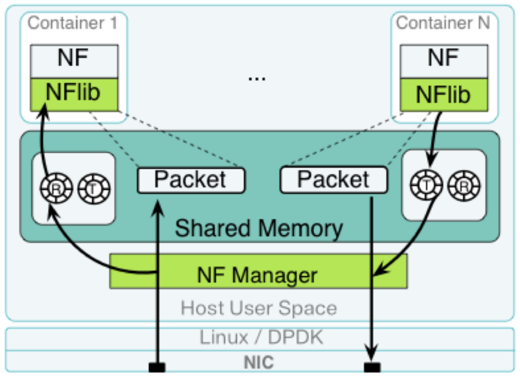

Fig. 1. Architecture of openNetVM platform

within a Docker container and associates to memory allocated by the manager. The manager is then responsible for utilizing the DPDK libraries to send and receive network traffic and route it between NFs. openNetVM can achieve all this while maintaining throughput of 10Gbps between NFs and physical nodes.

This type of networking platform introduces other challenges that restricts scalability. As more NFs are introduced to a network, more processing is required of the manager. Therefore, a structured communication protocol between each NF and the manager is required to avoid data loss and guarantee packet delivery.

Additionally, if the number of active NFs increases, we can assume that the network topology is more complex. These networks often have greater criticality due to the number of transactions needed to parse data. In such "always-on" networks, restarting the manager when starting or stopping an NF causes a period of data loss. This cannot happen; manager uptime must be prioritized.

NFs play a critical role in networks because they manipulate and analyze traffic. In order to maintain our performance and scalability achievements, openNetVM must provide ways to mitigate these issues.

III. Implementation

openNetVM addresses the challenges mentioned above by providing two features: NFlib, which provides a structured communication protocol between an NF and the manager; and the ability to dynamically manage NF clients as they are created or destroyed.

A. NFlib Guest Library

The NFlib Guest Library is composed of two smaller libraries. One is an API to handle communication between the manager and other NFs. The other is a packet helper library which exposes raw packet information from DPDK's packet wrapper.

The guest library is the main interface between an NF and the manager. It exposes two interfaces to each NF: the initialization sequence and the run sequence. The initialization sequence gives each NF the opportunity to provide information about itself to the manager. Afterwards, the NF is allowed to associate with the manager. Then, during the run sequence, the NF provides a packet handler function to the manager. Whenever the manager receives a packet destined for a certain NF, it simply executes the appropriate packet handler function. This interface creates an abstraction where each NF only receives one packet at a time. Limiting each NF's scope decreases the time needed to create a new NF since developers only have to focus on processing information. This also brings scalability to each NF since they can be moved throughout the network expecting the same amount of information each time.

Since we use DPDK to send and receive data from the network interface, we are initially limited by its packet abstraction. DPDK exposes the raw string of bytes that make up a packet, as a structure called *mbuf*. While it is possible to write NF's that solely interface with *mbuf*s, it is difficult to create complex NFs using this method. For example, Deep Packet Inspection (DPI) requires access to the TCP or UDP header fields in the packet. The packet helper library achieves this by providing an abstraction over DPDK's packet wrapper to expose the full packet headers and metadata in a consistent manner.

B. Dynamic NF Management

The openNetVM manager has the ability to dynamically register Network Functions as they start up. This means that the manager is able to see when an NF starts, allocate its memory, and initialize its necessary data structures. Now, network topologies can dynamically adjust when new NFs are spawned without any downtime.

This creates many opportunities for networks that were previously difficult or expensive to implement. For example, if there is a firewall NF that blocks some network traffic and the security administrator wants more information regarding the suspicious traffic, openNetVM provides the ability to spawn a DPI NF. The security administrator would then create an exception in the firewall NF to route the suspicious traffic into the DPI NF. Or, if the network's traffic load outgrows a single firewall's capacity, then a load balancer NF and additional firewall NFs can be spawned to distribute the traffic evenly. Traditionally, these actions would require disabling the network, physically moving and adding NFs, and restarting the network. With openNetVM's dynamic manager, these actions do not impact uptime.

The openNetVM Manager uses a shared memory region to store information about currently running NFs. When an NF starts, the NFlib guest library records identifiable information about the NF and enqueues it in this shared memory region. Then, the manager can determine if a new NF has started and assign it an internal ID. Once the NF is assigned an ID, it can start listening for and processing incoming packets.

Allowing networks to dynamically adjust to new NFs allows for more complex network topologies. Networks using openNetVM can maintain a high performance rate while remaining elastic enough to adjust to topology changes without any interaction or downtime.

IV. Conclusion

It is integral that enterprise and data center level networks provide a resilient and scalable environment to process data. NFs constantly need to be created and destroyed to provide the tools to analyze and manipulate network data. As trends in industry push networks to utilize NFV techniques, new performance and scalability problems arise. openNetVM solves these challenges by providing a NF platform that not only guarantees performance of 10Gbps, but also remains scalable with the NFlib library and the ability to dynamically adjust and manage new NFs starting and redundant NFs stopping.

V. References

[1] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. nf.io: A file system abstraction for nfv orchestration. In SIGCOMM, 2015.

[2] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In NSDI, 2012.

[3] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone elses problem: Network processing as a cloud service. SIGCOMM Comput. Commun. Rev., 2012.

[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone elses problem: Network processing as a cloud service. SIGCOMM Comput. Commun. Rev., 2012.

[5] VMware Inc. Best practices for performance tuning of telco and NFV workloads in vSphere. VMware Inc., 2015

[6] http://www.networkworld.com/article/2163183/data-breach/f5-introduces-new-integrated-firewall-capability-for-the-data-center.html

[7] http://newegg.com/Product/Product.aspx?Item=9SIA24G1XA6455

[8] http://www.bradreese.com/blog/7-21-2014.htm

[9] http://www.dell.com/us/business/p/poweredge-r920/pdf