nf.io: A File System Abstraction for NFV Orchestration

Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
{mfbari, sr2chowdhury, r5ahmed, rboutaba}@uwaterloo.ca

CCS Concepts

•**Networks** → **Network architectures; Network design principles;**

Keywords

Network Function Virtualization; Service Chain Orchestration; File System Abstraction

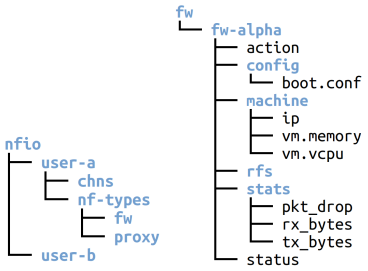
1. INTRODUCTION

Middleboxes have become an integral part of modern enterprise and data center networks. They are used for realizing various performance and security objectives. Most middlboxes (*e.g.*, firewalls, Intrusion Detection Systems (IDSs), Network Address Translators (NATs), *etc.*) are dedicated hardware appliances. However, recent advancements in cloud and virtualization technologies have fueled the concept of Virtual Middleboxes or Virtual Network Functions (VNFs) along with a new research field known as Network Function Virtualization (NFV). This area of research has gained a lot of traction from both industry and academia. Although much progress has been made in NFV technology, a crucial component for realizing the primary objective of NFV is still missing – a management and orchestration [7] system that conforms to the principles of NFV: open source, open API and standardized software solutions. Without this feature, network operators may end-up with the same situation of vendor lock-in as with proprietary hardware middleboxes. A number of recent proposals like Stratos [8], OpenNF [9], and Split/Merge [10], strive to fulfill the requirements for VNF management and orchestration. However, they propose incompatible northbound APIs. What is really needed is a standardized API that is flexible enough to express a wide range of NFV management and orchestration operations. History shows that standardization efforts usually take a long time and often are futile. Hence, we take a different approach, and propose to use an existing, well known, standardized interface for NFV management and orchestration: the Linux file system interface.

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790028



(a) `nf.io` root (b) VNF Instance

Figure 1: `nf.io` File System Abstraction

We call our proposed system `nf.io`. It utilizes the Linux file system as the northbound API for VNF orchestration. It adopts various operating system principles: (i) everything (resource, configuration) is represented as files, (ii) common Linux utility programs (*e.g.*, `mkdir`, `cp`, `mv`, `ln`, *etc.*) are used for state manipulation, (iii) heterogeneous resource pools (*e.g.*, different networking tool-chains like Linux bridge or Open vSwitch [5]) are controlled through a high-level abstraction, and (iv) resource specific drivers are developed similar to device drivers in an OS. Existing NFV management and orchestration systems like Stratos or OpenNF can use the `nf.io` abstraction by developing resource drivers specific to their requirements.

2. SYSTEM DESCRIPTION

2.1 Features

Key features of `nf.io` are as follows:

- **Everything is a file:** States and configurations of a VNF deployment are represented as files organized in a hierarchical directory structure.
- **Centralized control:** A centralized point of control over a distributed VNF deployment.
- **Compatibility:** A rich set of existing file system utilities (*e.g.*, `grep`, `mkdir`, *etc.*) and configuration management tools (*e.g.*, Chef, Puppet, *etc.*) can be used with `nf.io` for VNF management.

2.2 File System Abstraction

`nf.io` uses a simple and intuitive directory hierarchy to store states regarding VNF deployment, configuration and chaining. A high-level view of the `nf.io` directory hierarchy is shown in Figure 1. The root of the file system with two users is shown in Figure 1(a). The `user-a` and `user-b` directories mark the home directory for the users. The VNFs

and chains deployed by a user are organized under his home directory. The structure of a directory representing a VNF is shown in Figure 1(b). The `config` and `machine` directories contain configuration parameters. The `action` file is used by the user to perform different VNF operations (*e.g.*, `start`, `pause`, `resume`, `kill`, *etc.*). The `status` file indicated the current status of the VNF (*e.g.*, `running`, `paused`, `error`, *etc.*). The files contained under the `stats` directory are used to collect data like packet drops, transmitted/received bytes, *etc.* The `stats` directory contains one file for each measurement metric. The `rfs` directory mounts the file system of the VNF itself, so that the user can directly change a configuration file and also read different kinds of statistics from the VNF. A VNF chain is deployed by creating a directory under the `chns` directory. A chain directory contains symbolic links to the VNF instances that are part of the chain. It also contains markers to indicate the start and next VNFs in the chain.

2.3 Architecture

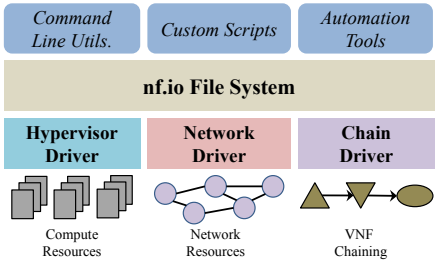


Figure 2: `nf.io` Architecture

A high-level view of the `nf.io` architecture is shown in Figure 2. The `nf.io` File System is a virtual file system that runs on top of the traditional OS file system. VNF operations are triggered when a user writes a operation string in the `action` files. `nf.io` performs these operations by using three resource drivers: (i) Hypervisor Driver, (ii) Network Driver, and (iii) Chain Driver.

2.3.1 Hypervisor Driver

In `nf.io`, network functions can be deployed in a number of ways. They can run as processes on a physical machine, VMs on a hypervisor like Xen or KVM, or as light-weigh containers provided by Docker [1] or Linux Container (LXC) [4]. The hypervisor driver abstracts the underlying diversity in these virtualization technologies and provides a uniform interface to `nf.io`.

2.3.2 Network Driver

`nf.io` requires support for certain networking functionality from the underlying physical infrastructure. In each physical machine, `nf.io` must have the ability to (i) setup bridges, (ii) create IP links between virtual ethernet (`veth`) pairs, (iii) setup tunnels (*e.g.*, VXLAN or GRE), and (iv) install forwarding rules. Similar to the hypervisor driver, the network driver hides the underlying heterogeneity and provides an abstract network interface to nf.io.

2.3.3 Chain Driver

The chain driver interconnects different types of VNFs. It provides a function `chn-cnct(vnf1, vnf2)`, where `vnf1` and `vnf2` are two arbitrary VNFs. For a chain like a → b → c, this function must be called twice: first for a → b, and again for b → c. The task of Interconnecting two VNFs depends primarily on their types, and whether their network interfaces are on the same or different IP subnets.

2.4 Implementation

The `nf.io` prototype is implemented using the python API binding for FUSE [2]. We rewrote a number of file system calls like `mkdir`, `read`, `write`, `symlink`, *etc.* to implement the `nf.io` file system semantics. The Hypervisor Driver currently supports KVM, Xen and Docker. We use libvirt [3] and Docker Remote API to control VMs and containers in KVM/Xen and Docker, respectively. The Network and Chain Drivers currently support two configurations: (i) Linux `iptables` and Linux bridge and (ii) Open vSwitch. In both cases we use GRE tunnels to connect VNFs deployed on different physical machines. Finally, we remotely mount the VNF's file system under the `rfs` directory (Figure 1) using `sshfs` [6]. A demonstration of `nf.io` is available at http://faizulbari.github.io/nf.io/.

3. DEMONSTRATION

We demonstrate the capabilities of `nf.io` by showcasing use cases focused on three primary areas: (i) configuration, (ii) deployment, and (iii) monitoring of VNF instances and chains. First, we will show how to configure different parameters of a single VNF instance. Then we will configure a service chain consisting of multiple VNFs and tweak different chain level parameters. Next, we will deploy the service chain on Docker containers and run a client to generate some test data. Finally, we will demonstrate `nf.io`'s monitoring features by querying data both at the VNF and chain levels.

4. ACKNOWLEDGMENTS

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

5. REFERENCES

[1] Docker. http://docker.com/.
[2] fusepy. https://github.com/terencehonles/fusepy.
[3] libvirt: The virtualization API. http://libvirt.org/.
[4] LXC: Linux Containers. https://linuxcontainers.org/.
[5] OVS: Open vSwitch. https://linuxcontainers.org/.
[6] sshfs. http://fuse.sourceforge.net/sshfs.html.
[7] Bari, M. F., Chowdhury, S. R., Ahmed, R., and Boutaba, R. On orchestrating virtual network functions in NFV. *CoRR abs/1503.06377* (2015).
[8] Gember, A., Krishnamurthy, A., John, S. S., Grandl, R., Gao, X., Anand, A., Benson, T., Akella, A., and Sekar, V. Stratos: A network-aware orchestration layer for middleboxes in the cloud. Tech. rep., 2013.
[9] Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S., and Akella, A. OpenNF: Enabling innovation in network function control. In *Proc. of SIGCOMM* (2014), ACM, pp. 163–174.
[10] Rajagopalan, S., Williams, D., Jamjoom, H., and Warfield, A. Split/merge: System support for elastic execution in virtual middleboxes. In *Proc. of USENIX NSDI* (2013), pp. 227–240.