Virtual Network Function Orchestration with Scylla

Roberto Riggio
CREATE-NET
Trento, Italy
rriggio@create-net.org

Julius Schulz–Zander
TU-Berlin
Berlin, Germany
julius@inet.tu-berlin.de

Abbas Bradai
CNRS-LiG
Grenoble, France
bradai@imag.fr

ABSTRACT

Network Function Virtualization promises to reduce the cost to deploy and to operate large networks by migrating various network functions from dedicated hardware appliances to software instances running on general purpose networking and computing platforms. In this paper we demonstrate *Scylla* a *Programmable Network Fabric* architecture for Enterprise WLANs. The framework supports basic Virtual Network Function lifecycle management functionalities such as instantiation, monitoring, and migration. We release the entire platform under a permissive license for academic use.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations

Keywords

Network Function Virtualization, Enterprise WLANs

1. INTRODUCTION

Network Function Virtualization (NFV) promises to reduce the cost to deploy and to operate large networks by migrating various network functions from dedicated hardware appliances to software instances running on general purpose networking and computing platforms. This in time shall improve the flexibility and the scalability of the network in that the deployment of new features and services can be quicker (software vs hardware life–cycles) and different network functions can share the same computing resources paving the way to further economies of scale.

Nevertheless, current NFV platforms typically only account for (virtualized) computing, storage, and networking resources with each of these resource managed separately. In this demo we want to take a step toward a truly deep programmable network by introducing the concept of *Programmable Network Fabric*. Our architecture leverages on a single platform consisting of general purpose hardware (x86) and operating system (Linux) in order to deliver three kinds of virtualized network resources, namely: basic forwarding nodes (i.e. OpenFlow–enabled Ethernet switches), packet processing nodes, and radio processing nodes (WiFi).

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

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

DOI: http://dx.doi.org/10.1145/2785956.2790040

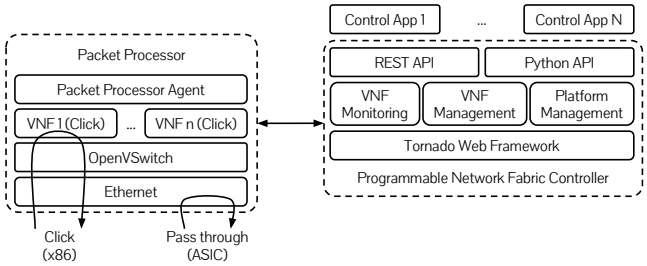

Figure 1: *Programmable Network Fabric Controller*.

During this demo we will show *Scylla* a *Programmable Network Fabric* architecture for Enterprise WLANs. *Scylla* supports basic Virtual Network Function (VNF) lifecycle management functionalities (instantiation, monitoring, and migration). The demo will demonstrate the framework operation using three sample VNFs: a performance enhancing VNF, a security VNF, and a monitoring VNF. Finally we release the code under a permissive license for academic use[1].

2. PROGRAMMABLE NETWORK FABRIC

We name *Programmable Network Fabric* the set of all packet processing nodes. As it can be seen in Fig. 1 each packet processing node includes an OpenVSwitch instance, one or more VNFs, and one Packet Processor Agent. The latter is in charge of monitoring the status of each VNF as well as handling CRUD requests coming from the *Programmable Network Fabric Controller* (*PNFC*). The monitoring features includes: number of packets/bytes transmitted and received as well as the amount of resources (cpu time, memory, and storage) utilized by each VNF.

The *PNFC* acts as an orchestrator deciding whether a particular VNF request can be accepted or if it must be refused [1]. If a request is accepted, then the *PNFC* is in charge of mapping the request onto the substrate network, i.e., network resources must be allocated and configured on the substrate nodes and links and the VNFs must be installed on the selected nodes.

A combination of frameworks is used in our architecture. POX [2] is used in order to configure resources in the switching fabric, while the SD–RAN controller proposed by the authors in [3] is used in the wireless access. Finally, we use Click [4] as a single solution for advanced packet processing. Click allows to build complex VNFs using simple and

[1]http://scylla.create-net.org/

reusable components, called *Elements*. Click includes over 300 elements supporting functions such as packet classification, access control, deep packet inspection. Elements can be composed in order to realize complex VNFs. Finally, Click is easily extensible with custom processing elements making it possible to support features that are not provided by the standard elements.

The *PNFC* is build on top of the Tornado Web Framework [5]. Communication between packet processors and the *PNFC* takes place over a persistent TCP connection. Control applications run on top of the *PNFC* and exploit its programming primitives through either a REST API or a native Python API.

Scylla VNFs are modular and reusable components consisting of a Click instance wrapped inside a Python object exposing lifecycle management functions. A *Manifest* is attached to each VNF specifying the Click configuration, the number of input/output ports as well as a list of events that can be subscribed by control applications. The latter allow programmers to trigger a callback the first time a certain condition is verified. For example the `cpu_util` primitive will trigger a callback the first time the CPU utilization of any Packet Processor in a tenant's network exceeds 70%:

```
cpu_util(relation='GT',
         value=0.7,
         tenant_id='<UUID>',
         callback=cpu_callback)
```

Listing 1: Create an CPU Utilization trigger.

After the trigger has fired the first time and as long as the CPU utilization remains above 70%, the callback method is not called again by the same packet processor however the same callback may be triggered by other packet processors.

Scylla VNFs may also declare a state in their *Manifest* file. A VNF state is defined in the form of Click handlers that must be invoked in order to obtain a snapshot of an active VNF. Such snapshots are atomic and lock the VNF execution for the time required to invoke the handlers. State handlers must support both read and write operations allowing a VNF state to be migrated from one node to another.

3. DEMO

In this demo we show the operation of the *Scylla* framework using three VNFs. This section describes the purpose of each VNF. During the demo a web–based dashboard will be used to both show real–time statistics (traffic, cpu load, memory usage, energy consumption) as well as to deploy and migrate VNFs.

3.1 Uplink/Downlink Decoupling

Wireless, and in particular mobile networks, have been so far designed around the requirements of the downlink. In the recent years, however, we have observed a growth of new uplink–centric applications such as Machine Type Communications and Internet of Things. This calls for a paradigm shift where the traffic originated from a wireless client is opportunistically received by multiple in–range APs. However, if not properly controlled such a feature can lead to an overload in the network core. For example, a wireless client scheduled on N APs in the uplink direction could increase the load on the network core by a factor of N. Moreover, a straightforward implementation of such a mechanism can lead to a significant increase of frames which in turn can trigger an unstable behavior at the transport layer.

In order to address this issue we implemented a VNF which filters out duplicate 802.11 frames based on their sequence number. Traffic originated from clients is received at one or more APs where it is encapsulated (802.11 over Ethernet) and then forwarded to a packet processing node where the frame filtering VNF is deployed. This VNF is also responsible for removing the 802.11 and the LLC headers and for encapsulating the frame into an Ethernet header before forwarding it to its intended destination. The Click script implementing this VNF is reported in the listing below[2].

```
FromHost(vnf0)
 -> in :: Counter
 -> Strip(14)
 -> dupe :: WifiDupeFilter()
 -> decap :: WifiDecap()
 -> out :: Counter
 -> ToHost(vnf0);
```

Listing 2: Duplicate filtering VNF.

3.2 Firewall Migration

Today's enterprise networks often need to deal with BYOD where employees or customers connect their own devices to a corporate network. This, however, raises new requirements on the network access policies. Our second VNF implements a straightforward firewall where rules can be moved from one instance to another depending on the client association state, i.e., rules are migrated when clients perform a handoff from one AP to another. This firewall VNF takes a list of tcpdump–like patterns as input from the controller and applies them to the incoming traffic.

3.3 SLA Monitoring

This VNF aims at implementing a basic SLA monitoring solution. In this scenario a packet sniffing VNF is deployed on radio nodes. This VNF collects all transmissions within decoding range of the radio node. For each link–layer event the following meta–data is tracked: *RSSI* (in dB), *Transmission Rate* (in Mb/s), *Length* (in bytes), and *Duration* (in μsec). The collected meta–data is then forwarded to another VNF which computes aggregate statistics by filtering our duplicate frames. In particular for each active WiFi station the number of packets/bytes transmitted and received as well as the retransmission count are tracked.

4. REFERENCES

[1] R. Riggio, T. Rasheed, and R. Narayanan, "Virtual network functions orchestration in enterprise wlans," in *Proc. of IEEE ManFI*, Ottawa, ON, Canada, 2015.

[2] "POX." [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[3] R. Riggio, M. Marina, J. Schulz-Zander, S. Kuklinski, and T. Rasheed, "Programming abstractions for software-defined wireless networks," *Network and Service Management, IEEE Transactions on*, vol. 12, no. 2, pp. 146–162, June 2015.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[5] "Tornado Web Server." [Online]. Available: http://www.tornadoweb.org/

[2] `vnf0` is a virtual interface attached to the OpenVSwitch instance running on the packet processing node.