

Managing NFV using SDN and Control Theory

Nabeel Akhtar Ibrahim Matta Yuefeng Wang
Computer Science Department, Boston University
Boston, MA 02215
{nabeel, matta, wyf}@bu.edu

Abstract—Control theory and SDN (Software Defined Networking) are key components for NFV (Network Function Virtualization) deployment. However little has been done to use a control-theoretic approach for SDN and NFV management. In this paper, we describe a use case for NFV management using control theory and SDN. We use the management architecture of RINA (a clean-slate Recursive InterNetwork Architecture) to manage Virtual Network Function (VNF) instances over the GENI testbed. We deploy Snort, an Intrusion Detection System (IDS) as the VNF. Our network topology has source and destination hosts, multiple IDSes, an Open vSwitch (OVS) and an OpenFlow controller.

A distributed management application running on RINA measures the state of the VNF instances and communicates this information to a Proportional Integral (PI) controller, which then provides load balancing information to the OpenFlow controller. The latter controller in turn updates traffic flow forwarding rules on the OVS switch, thus balancing load across the VNF instances.

This paper demonstrates the benefits of using such a control-theoretic load balancing approach and the RINA management architecture in virtualized environments for NFV management. It also illustrates that GENI can easily support a wide range of SDN and NFV related experiments.

I. INTRODUCTION

Network Function Virtualization (NFV) aims to implement network functions (*e.g.*, firewalling, NAT, intrusion detection system) as software instead of dedicated physical devices (middleboxes). NFV aims to decouple the network functions from proprietary devices, and it is designed to virtualize and consolidate network functions onto industry standard high-volume servers, switches and storage. Advantages of NFV include reducing equipment cost, speeding up new service deployment, and achieving high service performance [1], [2].

NFV management is the core task in NFV deployment and different frameworks have been proposed for NFV management. NFV management can be broadly classified into two components: (1) *NFV elastic management*, and (2) *NFV service chain management*.

A. *NFV elastic management*

NFV elastic management includes tasks related to Virtual Network Function (VNF) stateful migration from one Virtual Machine (VM) to another, and adding or removing VNF instances depending on the load on the system [3], [4], [5].

B. *NFV service chain management*

NFV service chain management deals with network traffic steering policies which dictate how traffic in the network should traverse a sequence of middleboxes (VNFs) [6], [7],

[8], [9], *e.g.*, to satisfy access control (ACL) policies specified by the network administrator.

NFV elastic management has recently received considerable attention in the research community [3], [4], [5]. However, most of this work focuses on VNF stateful migration, with the assumption that VNF state information (*e.g.*, load on VMs) is available across the system. In this paper, we use a new internet architecture – the Recursive InterNetwork Architecture (RINA) [10], [11], [12], [13] – to share VNF state information across the system and use a control-theoretic approach for managing load across VNF instances. To the best of our knowledge, this is the first work that uses a control-theoretic approach to NFV management.

Specifically, we describe a use case of managing Snort [14], an Intrusion Detection System (IDS) as the VNF. Our network topology on the GENI (Global Environment for Network Innovations) testbed [15] has source and destination hosts, multiple IDSes, an Open vSwitch (OVS) and an OpenFlow controller. A distributed management application running on RINA measures the state of the VNF (IDS) instances and communicates this information to a Proportional Integral (PI) controller [16], which then provides load balancing information to the OpenFlow controller. The latter controller in turn updates traffic flow forwarding rules on the OVS switch, thus balancing load across the VNF instances.

The rest of the paper is organized as follows. Section II provides an overview of the system: the GENI testbed, Snort IDS as the VNF, RINA application management architecture, the PI control, and the PI-based OVS controller. Section III describes our experimental setup, and Section IV presents our results. Section V concludes the paper with a summary and future work.

This work has recently been demonstrated at the IEEE ICNP NICE workshop [17].

II. SYSTEM OVERVIEW

Figure 1 shows an overview of the system. We deploy Snort [14], an Intrusion Detection System (IDS) as the VNF. There can be multiple source and destination hosts and all traffic directed from any source to any destination passes through Snort-IDS. VNF hosts run a distributed monitoring application (deployed over RINA) where each application instance shares the state of the VNF (*i.e.*, load information) with the central controller. The controller runs a control-theoretic Proportional Integral (PI) control algorithm that balances load across the VNF instances by providing the OVS controller with load balancing information, which is then used to update the flow

forwarding rules on the OVS switch so new flows are directed to less loaded VNF instances.

We next explain the system in more detail.

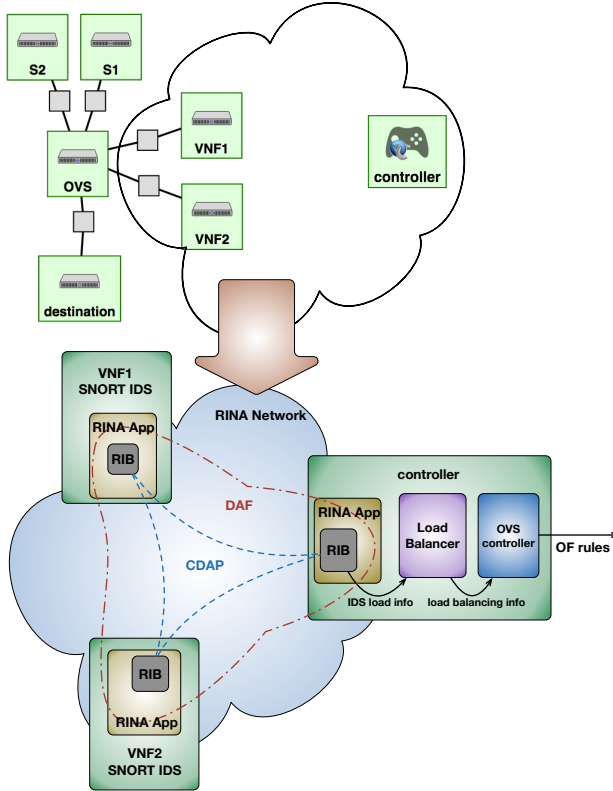


Fig. 1. System overview

A. GENI testbed

GENI (Global Environment for Network Innovations) [15] is a nationwide suite of infrastructure that enables research and education in networking and distributed systems. GENI supports large-scale experimentation with advanced protocols for data-centers, clouds, mobile and SDN networks, *etc.*

As shown in Figure 2, we reserved VMs on the GENI testbed for our use case. VMs are connected to each other through an OVS switch using layer-3 (IP level) connectivity. We have five VMs that host two VNF instances of Snort-IDS (VNF1 and VNF2), two sources (S1 and S2) and a single destination. We also have a controller VM that runs the PI and OVS controllers, along with a monitoring application instance (deployed over RINA) to collect VNF load information from other application instances running on the VNF VMs.

As shown in Figure 2, we reserved two sets of GENI resources on two different GENI slivers. A GENI sliver is one or more resources provided by a GENI aggregate [18]. The *Network Sliver* contains VMs for the sources, destination, OVS switch and VNFs, while the *Controller Sliver* contains

the controller VM. Resources in GENI are described in XML files called Resource Specifications (RSPecs). Our RSpec files for the *Network Sliver* and *Controller Sliver* can be found at [19] and can be used to reproduce our topology.

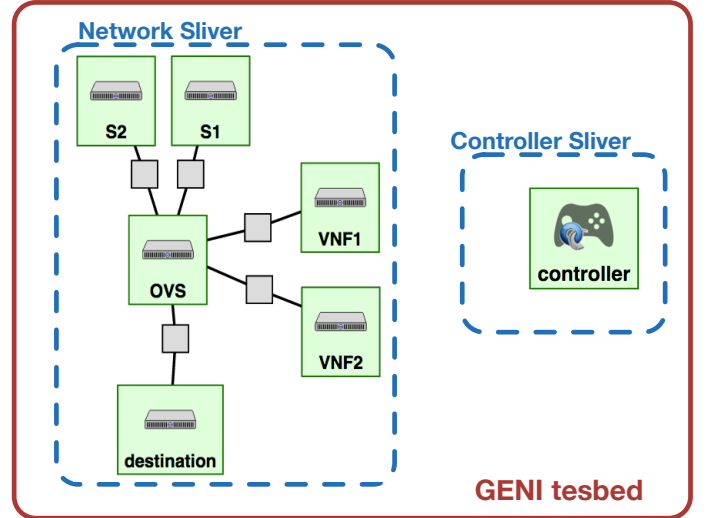


Fig. 2. GENI testbed

B. Snort-IDS as VNF

Snort-IDS [14] is an open-source network intrusion detection and prevention system (IDPS). It has the ability to perform real-time traffic analysis on IP networks. It is one of the most widely deployed IDPS.

We installed Snort on each VNF VM. Detailed instructions on installing Snort are given in [20]. We ran Snort in IDS mode to analyze traffic against the open-source Snort community rule set [14]. Snort community rules refer to all the rules that have been submitted by members of the open-source community. These rules are freely available to all Snort users and they are updated daily.

In our system, all traffic directed from any of the sources to the destination is checked against these rule sets for intrusion detection. This generates load on the VNF instances and this load information is provided to the *controller* using the RINA-based distributed monitoring application, which we explain next.

C. RINA

The Recursive InterNetwork Architecture (RINA) [10], [11], [12], [13] is a clean-slate network architecture that overcomes inherent weaknesses of the current internet, *e.g.*, security and support for mobility and quality of service. RINA is based on the fundamental principle that *networking is Inter-Process Communication (IPC) and only IPC*. RINA has two main design principles: (1) divide and conquer (recursion), and (2) separation of mechanisms and policies. Figure 3 shows an overview of the RINA architecture.

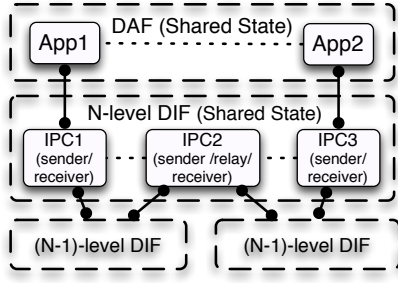


Fig. 3. RINA overview

1) *Distributed Application Facility*: A Distributed Application Facility (DAF) is a collection of distributed application processes that work together, using shared states, to perform a certain task, *e.g.*, video streaming or health analytics service. A Distributed IPC Facility (DIF) is a special DAF, where application processes are specialized to provide IPC, *i.e.*, they cooperate to provide communication service over a certain scope. As shown in Figure 3, a higher-level DIF provides communication service over a larger scope by recursively using smaller scope communication services provided by lower-level DIFs.

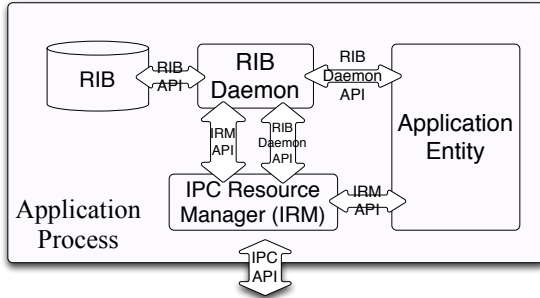


Fig. 4. Application Process Components and RINA APIs

2) *RINA Application Management*: Figure 4 shows the key components of an application process in RINA. The *Resource Information Base (RIB)* is the database that stores all application process information. The *RIB Daemon* helps other components of the application process access information stored in the local RIB or in a remote application's RIB. In the latter case, an object-based protocol, called *Common Distributed Application Protocol (CDAP)*, is used to access remote management objects. The *IPC Resource Manager (IRM)* manages the use of underlying IPC processes belonging to low-level DIFs that provide communication services to this application process with other application processes.

RIB Daemon API
public int createEvent(SubscriptionEvent subscriptionEvent);
public void deleteEvent(int subscriptionID);
public Object readSub(int subID);
public void writePub(int pubID, byte[] obj);
IRM API
public int allocateFlow(Flow flow);
public void deallocateFlow(int handleID);
public void send(int handleID, byte[] msg) throws Exception;
public byte[] receive(int handleID);

TABLE I
RIB Daemon API and IRM API

As shown in Table I, RINA provides two sets of APIs: (1) *RIB Daemon API*, and (2) *IRM API*, for the users to write applications and to support new network management policies. The RIB Daemon API is based on a publish/subscribe model, which supports publishing information through the *Pub* event and retrieval of information using the *Sub* event. The IRM API supports the allocation and deallocation of application connections (flows) to other application processes, and the sending and receiving of messages over existing connections (flows). More details about the RINA APIs can be found in [12], [21].

For our system, we created a RINA monitoring DAF (consisting of monitoring application processes) that monitors the VNF instances. The controller uses this DAF to get the state (load information) of the VNF instances. The RIB Daemon API is used to exchange application monitoring objects that store the state (load) of each VNF instance. Each monitoring application process on the VNF VMs periodically publishes its VNF load information. The application process running on the controller VM subscribes to this information, and passes the average over the last few measurements to the PI controller for load balancing, as explained next. In this paper, the load information is published every 0.5 second and the averaging is done over the last 10 measurements.

D. PI Controller

The block diagram of the Proportional Integral (PI) controlled NFV system is shown in Figure 5. The RINA-based distributed monitoring application provides the VNF state (load) information $L(t)$ to the PI controller. We assume that initially, all traffic is directed to a single VNF (Snort IDS) instance, VNF1, and $L(t)$ represents the current average CPU load on VNF1. Given a target CPU load T , representing the maximum capacity of a VNF instance, if the load on VNF1 exceeds T , new traffic flows are forwarded to a second VNF instance, VNF2. Assuming instantaneous feedback / measured load $L(t)$, the PI control equation is given by:

$$x(t) = \max[0, \min[1, x(t-1) + K(\frac{L(t)}{T} - 1)]] \quad (1)$$

where $x(t)$ is the ratio of traffic diverted to VNF2 at time t , and K is the controller's gain.

Observe that if the load on VNF1 is less than the target load T , the system converges to a state where no traffic is forwarded

to VNF2, *i.e.*, $x(t) \rightarrow 0$ and only one VNF instance (VNF1) is used under light loading conditions. On the other hand, if the load on VNF1 is higher than the target load T , then the system converges to a state where new traffic flows are forwarded to VNF2 as $x(t) \rightarrow 1$.

The block diagram in Figure 5 is obtained by taking the Laplace transform of the continuous unconstrained version of the dynamic equation (1):

$$\frac{dx(t)}{dt} = K'(L(t) - T) \quad (2)$$

Taking the Laplace transform, assuming $x(0) = 0$ and a constant target T , yields:

$$sX = K'(L - \frac{T}{s}) \quad (3)$$

where X and L represent the Laplace transforms of the corresponding functions in the time domain, *i.e.*, $x(t)$ and $L(t)$, the Laplace transform of the target $T(s) = \frac{T}{s}$, $K' = \frac{K}{T}$ is the controller's gain, and s is a complex variable. The PI control block $\frac{K'}{s}$ takes as input the error, which is the difference between the load and target ($L - \frac{T}{s}$), and produces the control signal X , which is proportional to the error (K' is the proportionality factor) and to the integral of the error (manifested by $\frac{1}{s}$) [16].¹

Algorithm 1 shows the implementation details of the PI controller. VNF load information is read from a text file $IDS_{load}.txt$ (line 6). The control variable $x(t)$ is calculated (line 7) and this information is saved (line 8) so that the OVS controller can later access it.

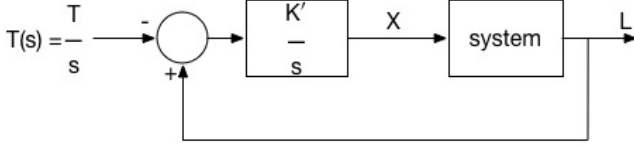


Fig. 5. Block diagram of the PI-controlled NFV system. System load L and target load $T(s) = \frac{T}{s}$ of VNF1 is used to compute X , *i.e.* ratio of traffic diverted to VNF2

Algorithm 1 PI controller

Input: $IDS_{load}.txt$

Output: $x(t)$

```

1:  $T = 0.5$ 
2:  $x(t-1) = 0.0$ 
3:  $x(t) = 0.0$ 
4:  $K = 0.2$ 
5: while True do
6:    $L(t) = \text{getLoad}(IDS_{load}.txt);$ 
7:    $x(t) = \max[0, \min[1, x(t-1) + K(\frac{L(t)}{T} - 1)]];$ 
8:    $\text{write}(t, x(t));$ 
9: end while

```

¹Recall that in the Laplace domain, differentiation becomes a multiplication by s , and integration becomes a division by s .

E. OVS Controller

1) *PI-based OVS controller:* Information about load balancing is provided to OVS by the PI controller. The control variable $x(t)$ provided by PI controller is the ratio of traffic diverted to a second VNF (Snort) instance. The OVS controller updates the forwarding rules on the OVS switch based on this information to bring the system to a more balanced steady state. Algorithm 2 provides the implementation details for the PI-based OVS controller. For each input flow f , the controller generates a random number between 0 and 1 (line 2). If the random number is greater than $x(t)$ (line 3), traffic of flow f is sent to IDS1 (line 4) for intrusion detection. Otherwise, traffic of flow f is forwarded to IDS2 (line 6).

A flow is defined by the 5-tuple: protocol type, and source and destination IP and port numbers. The OVS switch maintains two per-flow timers: an idle timer and a hard timer. The *idle timer* defines the time (in seconds) after which a flow entry is removed if no packets from that flow are observed. The *hard timer* defines the time (in seconds) after which a flow entry is removed regardless of whether packets from that flow have been observed. A flow is considered “new” if it has no flow entry on the OVS switch, and in this case, the controller directs its traffic based on the variable $vnfSelected$ (line 8). In this paper, we set both timers to 1 second.

Algorithm 2 OVS controller based on PI control

Input: $Flows, x(t)$

```

1: for all  $f$  in  $Flows$  do
2:    $random = \text{generateRandom}();$ 
3:   if  $random > x(t)$  then
4:      $vnfSelected = IDS1;$ 
5:   else
6:      $vnfSelected = IDS2;$ 
7:   end if
8:    $\text{sendFlow}(f, vnfSelected);$ 
9: end for

```

2) *Round Robin based OVS controller:* We also implemented an OVS controller based on a Round Robin (RR) load balancer. The RR balancer does not take load into account. The OVS controller running RR directs each new flow request to one of the two VNF instances in a round robin fashion. Algorithm 3 shows the implementation details for the RR-based load balancer.

Our controller files can be found at [19].

III. EXPERIMENTAL SETUP

A. Starting Snort

First Snort IDS is started on each VNF VM (host). Since each VM is multihomed, Snort IDS is started on the network interface that will receive traffic.

B. Creating a RINA monitoring DAF

A RINA monitoring DAF is created to share the load information of each VNF instance with the central controller. Each application instance publishes the load information to which the application instance on the controller host subscribes.

Algorithm 3 OVS controller based on RR load balancer

Input: *Flows*

```
1: vnfSelected = IDS1
2: for all f in Flows do
3:   if vnfSelected == IDS1 then
4:     vnfSelected = IDS2;
5:   else
6:     vnfSelected = IDS1;
7:   end if
8:   sendFlow(f, vnfSelected);
9: end for
```

C. Starting PI controller

The PI controller is now started. It reads the load information of the VNF instances provided by the RINA monitoring DAF and updates the control variable $x(t)$, the ratio of traffic directed to a second VNF instance (IDS2), according to (1).

D. Starting OVS controller

The OVS controller is started next. It uses the output from the PI controller (i.e., $x(t)$) to divert excess traffic from IDS1 to IDS2.

E. Traffic Generation

TCP traffic is generated using the iPerf application [22]. iPerf is a network bandwidth measurement tool that can generate either TCP or UDP traffic. We varied the number of iPerf instances to change the amount of traffic load on the VNF instances.

IV. RESULTS

We compare the PI-based load balancer with the traditional Round Robin (RR) load balancer. The PI controller takes into account the load on the VNF instances for load balancing while the RR load balancer directs individual flow requests to the VNF instances in a load-independent round robin fashion.

Figure 6 shows the CPU load on each VNF instance under the simple RR load balancer. Since RR does not take into account load on the VNF instances, one VNF instance might become overloaded compared to the other one. There can be multiple reasons for the load imbalance, e.g., some flows impose more load on the system than other flows, flows last for varying amount of time, or there can be load generated by other applications, besides the Snort application, running on the VNF host.

As shown in Figure 6, the VNF host running Snort-IDS-1 initially has around 38% CPU usage because of other applications running on it. We generate traffic at time 39 using iPerf with similar flows that last for the same amount of time. We can see that RR directs iPerf traffic equally between IDS-1 and IDS-2. However this overloads the first VNF instance (IDS-1) since there are other applications running on it.

Figure 7 shows the CPU load on each VNF instance under the PI-based load balancer. The target load (T) on IDS-1 is set to 50%. When we generate iPerf traffic at time 39, the load on IDS-1 increases beyond 50% and so new flows get

diverted to the second VNF instance (IDS-2). We can see that the system stabilizes soon after at an average load on IDS-1 of 50% while the rest of the load is diverted to IDS-2.

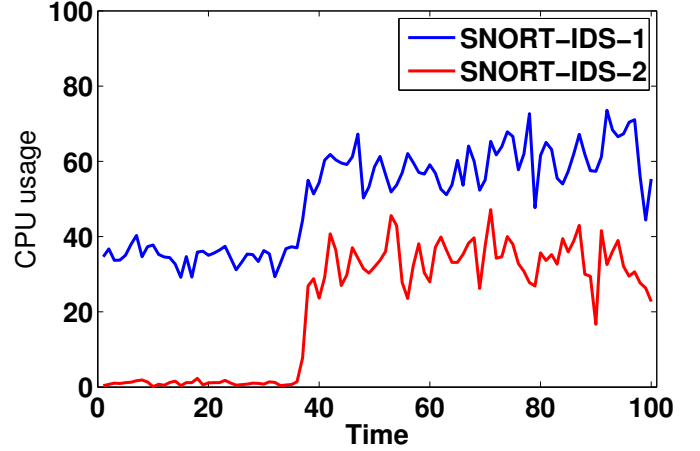


Fig. 6. Simple Round Robin load balancing

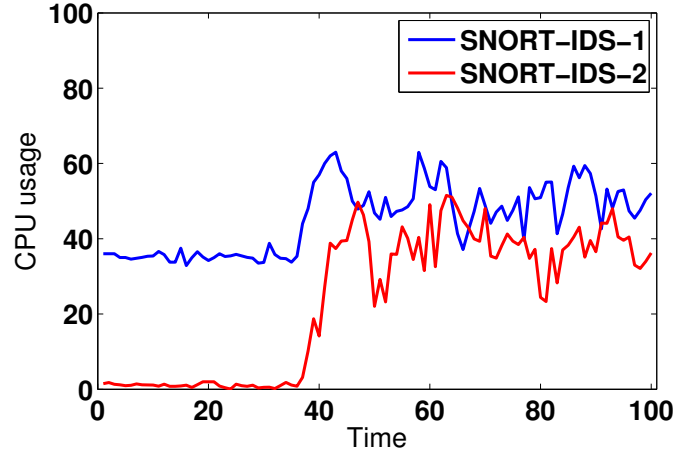


Fig. 7. Load balancing based on PI control ($T = 50\%$)

V. CONCLUSION

In this paper, we show how control theory can be used to manage NFV using SDN. We also show that the management architecture of RINA provides several facilities that can be used for easy NFV elastic management.

The GENI testbed is used for experimentation. GENI provides state of the art experimentation facility and can support a wide range of experiments, including NFV and SDN related experiments. The widely used Snort IDS is deployed as the VNF and traffic is directed to different Snort IDS instances for processing. Benefits of using a control-theoretic load balancing approach over a traditional Round Robin based load balancer are highlighted in this work.

We believe that SDN and control theory are key components for NFV deployment and much future work remains to be done. For example, one could consider other control objectives and controllers (*e.g.*, a PID controller [16]), other than the PI controller considered in this paper. Furthermore, one could experiment with larger networks and study the effect of the feedback (measurement) delay on the stability and performance of the NFV system. Control-theoretic convergence and stability analysis can be used to determine the maximum scope (*i.e.*, region of operation and performance characteristics) over which VNFs can be effectively deployed and managed (related to the concept of “fog computing / networking”). Other future work includes investigating VNF state migration and the interaction with VNF load balancing.

REFERENCES

- [1] ETSI, “Network Functions Virtualization (NFV) - White Paper,” https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf.
- [2] ETSI Network Functions Virtualization Industry Specification Group (NFV ISG), <https://portal.etsi.org/TBsiteMap/NFV/NFVMembership.aspx>.
- [3] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “OpenNF: Enabling Innovation in Network Function Control,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM ’14. New York, NY, USA: ACM, 2014, pp. 163–174. [Online]. Available: <http://doi.acm.org/10.1145/2619239.2626313>
- [4] Shriram, Rajagopalan, Dan, Williams, Hani, and Jamjoom, “Pico Replication: A High Availability Framework for Middleboxes,” in *ACM Symposium on Cloud Computing (SoCC)*, Santa Clara, California, Oct 2013.
- [5] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, “Split/Merge: System Support for Elastic Execution in Virtual Middleboxes,” in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi’13. Berkeley, CA, USA: USENIX Association, 2013, pp. 227–240. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2482626.2482649>
- [6] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, “PGA: Using Graphs to Express and Automatically Reconcile Network Policies,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM ’15. New York, NY, USA: ACM, 2015, pp. 29–42. [Online]. Available: <http://doi.acm.org/10.1145/2785956.2787506>
- [7] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, “Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags,” in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 543–546. [Online]. Available: <https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/fayazbakhsh>
- [8] Open Network Operating System (ONOS) Intent Framework, <https://wiki.onosproject.org/display/ONOS/The+Intent+Framework>.
- [9] SELinux, http://selinuxproject.org/page/Main_Page.
- [10] J. Day, I. Matta, and K. Mattar, “Networking is IPC: A Guiding Principle to a Better Internet,” in *Proceedings of ReArch’08 - Re-Architecting the Internet (co-located with CoNEXT)*, New York, NY, USA, 2008.
- [11] Boston University RINA Lab, <http://csr.bu.edu/rina/>.
- [12] Y. Wang, I. Matta, F. Esposito, and J. Day, “Introducing ProtoRINA: A Prototype for Programming Recursive-Networking Policies,” *ACM SIGCOMM Computer Communication Review (CCR)*, July 2014.
- [13] Y. Wang, I. Matta, and N. Akhtar, “Application-Driven Network Management with ProtoRINA,” in *Technical Report BUCS-TR-2015-003, Boston University*, March 2015, to appear in the Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2016), April 2016.
- [14] SNORT, <https://www.snort.org/>.
- [15] GENI, <http://www.geni.net/>.
- [16] I. Matta, “Optimizing and Modeling Dynamics in Networks,” in *eBook on Recent Advances in Networking*, H. Haddadi and O. Bonaventure, Eds. ACM SIGCOMM, August 2013, vol. 1, licensed under a CC-BY-SA Creative Commons license. [Online]. Available: http://sigcomm.org/education/ebook/SIGCOMMBook2013v1_chapter4.pdf
- [17] N. Akhtar, I. Matta, and Y. Wang, “Managing NFV using SDN and Control Theory,” <http://geni.net/nice2015/>, Nov. 2015.
- [18] GENI Glossary, <http://groups.geni.net/geni/wiki/GENIGlossary>.
- [19] GENI RSpecs, Load Balancer and Controller code, <http://csr.bu.edu/NFV>.
- [20] SNORT Installation Manual, <https://www.snort.org/documents/snort-users-manual>.
- [21] Y. Wang, F. Esposito, I. Matta, and J. Day, “Recursive InterNetworking Architecture (RINA) Boston University Prototype Programming Manual,” in *Technical Report BUCS-TR-2013-013, Boston University*, 2013.
- [22] iPerf, <https://iperf.fr/>.