Expressing and Managing Network Policies for Emerging HPC Systems

Sergio Rivera
University of Kentucky
Lexington, Kentucky
sergio@netlab.uky.edu

James Griffioen
University of Kentucky
Lexington, Kentucky
griff@netlab.uky.edu

Zongming Fei
University of Kentucky
Lexington, Kentucky
fei@netlab.uky.edu

Jane Huffman Hayes
University of Kentucky
Lexington, Kentucky
hayes@cs.uky.edu

ABSTRACT

Traditional high performance computing (HPC) centers that operate a single large supercomputer cluster have not required sophisticated mechanisms to manage and enforce network policies. Recently, HPC centers have expanded to support a wide range of computational infrastructure, such as OpenStack-based private clouds and Ceph object stores, each with its own unique characteristics and network security requirements. Network security policies are becoming more complex and harder to manage. To address the challenge, this paper explores ways to define and manage the new network policies required by emerging HPC systems. As the first step, we identify the new types of policies that are required and the technical capabilities needed to support them. We present example policies and discuss ways to implement those policies using emerging programmable networks and intent-based networks. We describe our initial work toward automatically converting human readable network policies into network configurations and programmable network controllers that implement those policies using business rule management systems.

CCS CONCEPTS

• **Networks → Network management**; *Programmable networks.*

KEYWORDS

Network Policies, Software-Defined Networks, HPC Systems

ACM Reference Format:
Sergio Rivera, James Griffioen, Zongming Fei, and Jane Huffman Hayes. 2019. Expressing and Managing Network Policies for Emerging HPC Systems. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3332186.3333045

1 INTRODUCTION

Any institution that operates a high performance computing (HPC) center understands the importance of securing the network and systems that comprise the HPC system. The level of security that must be maintained depends on the institution's security policies. Historically, many institutions have defined relatively simple security policies for their HPC systems, policies that can usually be implemented in straightforward ways. Generally speaking, users are given accounts on the system and must authenticate themselves using conventional methods (e.g., loginID/password or some sort of two factor authentication). While a secure login/authorization mechanism provides the base level security, most HPC systems look to the network to provide additional levels of protection – particularly from outside attacks – preventing network traffic that is not policy compliant from traversing HPC networks. Institutions will typically use a combination of firewalls and virtual private network (VPN) technologies to ensure that only authorized users are able to access the HPC network. In other words, users must first login to the VPN network (or be connected to an internal campus network that is allowed to access the HPC network), before they can even attempt to login to the HPC computing system (see Figure 1).

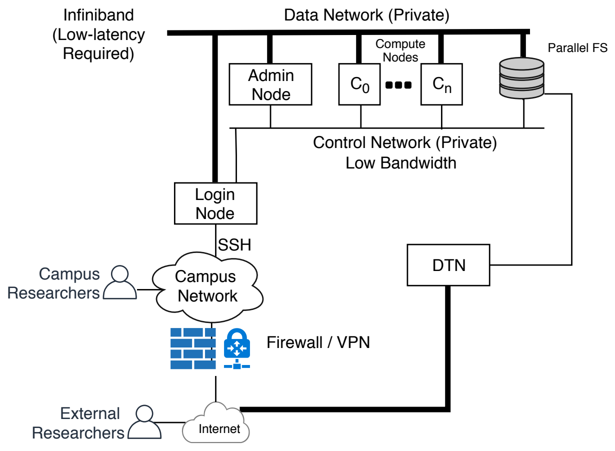


Figure 1: A typical HPC architecture

For example, a typical network security policy defined to protect a campus' HPC system might be *"Access to the HPC network is* *only allowed from machines on the campus network. To access the HPC network from off campus, users must login to the campus VPN service"*. This policy is relatively easy to implement by assigning internal (i.e., campus route-able) IP addresses to the HPC network and setting up a VPN/Firewall at the Internet edge that does not allow any incoming network traffic to the HPC network (except for traffic coming in over an authenticated VPN connection). In general, simple network policies like this have been sufficient to protect HPC center supercomputer systems. Because the user workflow involves logging in (via ssh) to the supercomputer and running (self-contained) jobs on the cluster, a simple network policy that only allows (ssh) remote login traffic to/from login nodes is sufficient and easy to implement.

Unfortunately, simple policies based on VPN access can pose a significant bottleneck to high-speed data transfers. As a result, many HPC centers have modified their network policy to allow for a Data Transfer Node (DTN) located outside the campus firewalls in a Science DMZ [9]. The DTN node allows high-speed external access to the HPC data storage system. In other words, the network policy has been enhanced to include *"To transfer data at high speed from outside campus, users can transfer data to/from the DTN node"*. This addition to the policy can be implemented in a straightforward manner by directly connecting the DTN node to the storage network. However, because the DTN is not protected by a firewall and/or intrusion detection/prevention system (IDS/IDP), it is susceptible to attacks. Consequently, the DTN node and OS must be "locked down" to the greatest extent possible and continuously monitored for possible attacks to ensure the HPC storage system is not compromised.

In recent years, HPC centers have begun expanding their services beyond just operating a large supercomputer cluster. In particular, they have started to support a wide range of computational infrastructure each with its own unique characteristics and network security requirements. As a result, network security policies are becoming more complex and require more advanced tools and mechanisms to define and manage the network security policy. For example, private clouds based on technologies such as OpenStack [20] are becoming a common place. These systems not only require network policies for the underlying hardware, but also for the virtual systems that run on the OpenStack cloud. Because they offer many different access methods and protocols, the network policies must define which ones are, and are not, allowed (e.g., VNC to VMs is allowed, but RDP is not allowed; or ssh is only allowed through network proxies). In addition, storage systems have become complex systems in and of themselves, consisting of multiple networks each with their own network policies. Specialized frameworks that support stream processing, map-reduce, time series database, and other forms of distributed processing [2] are also being deployed, each requiring its own set of network polices. Even HPC resources are evolving with the emergence of new control software designed to leverage advances in containerization. In short, these emerging environments require more advanced, finer-grained, and dynamic network policies that make it more difficult for system administrators to manage the network security of the system.

As an example, consider a Ceph object store [6] which can be accessed via an S3 interface, a block device interface, or a file system interface. HPC administrators may want to authorize some (privileged) machines on the campus network to be able to use the file system interface, or they may want to allow an OpenStack system to be able to use the block storage interface. Perhaps they would like to allow any user anywhere (inside or outside the campus network) to be able to access the S3 interface. Depending on the desired security policy, the network can be configured to ensure only policy compliant traffic is allowed on the network. Similarly, consider an OpenStack system where the system administrators want to prohibit VNC [21] access to VMs, and instead want to only support encrypted remote access protocols like the NX protocol [19] used by NoMachine [18]. In this case, the network should block all VNC traffic while allowing NX traffic to OpenStack VMs. While mechanisms may exist within the operating system or OpenStack to block these types of traffic, the goal of a network policy is to prevent unwanted traffic from entering or traversing the network at all. This becomes particularly important when users have the ability to modify the operating system or OpenStack settings – one of the "features" of cloud technologies. In short, emerging technologies require more complex network policies and more sophisticated network infrastructure to implement those policies.

To address these challenges, we have been exploring ways to define and manage the new network policies required by emerging HPC systems. A first step in this process is understanding the new types of policies that are required and what technical capabilities will be required to support them. Having identified the types of policies that are needed, we have also begun to look at ways for HPC system administrators to define policies and effectively manage the complex set of policies needed to operate an HPC system.

In this paper, we identify features of networks found in new types of computational systems being deployed by HPC centers, and discuss the types of network policies needed to manage these networks. We present example policies and discuss ways to implement those policies using emerging programmable networks and intent-based networks. We also briefly discuss our initial work toward automatically converting human readable network policies into network configurations and programmable network controllers that implement those policies using business rule management systems (BRMS).

The paper is organized as follows. Section 2 discusses the technologies used to dynamically enforce network policies in research environments. We give examples to show security policies for HPC systems are more complicated in Section 3. Section 4 presents our proposed method for specifying network policies for HPC systems and enforcing the policies by automatically translating these policy statements into SDN rules implemented in switches. Lastly, Section 5 concludes the paper.

2 MANAGING NETWORK POLICIES

The increasing complexity of HPC resources and the need for finer-grained protection calls for more advanced network policy mechanisms. The following briefly discusses recent advances in network management capabilities that can be used to implement the types of complex network management policies require by today's HPC systems.

2.1 Intent-based Networks

Recent advances in machine learning combined with the ability to collect and process large amounts of network configuration and telemetry data has given rise to so-called *intent-based networking (IBN)* systems [5, 10]. The idea behind intent-based networking is that network administrators should only have to specify what they want the network to do or how they want the network to be protected and then let the network figure out how to best configure the underlying networking devices to achieve that intent. Several mainstream as well as startup network vendors now support some form of IBN including Cisco [7], Juniper [14], Big Switch [4], Apstra [3], and others [17].

Sometimes referred to as *self-driving networks*, these networks are all based on the concept of a (logically) centralized control system (we will call it a "controller" but each vendor has its own terminology) that interacts with and controls all routers and switches in the network. These controllers monitor the network, collecting device configurations and state, as well as performance information, traffic logs, etc, on which they can run analytics software or learning algorithms to understand the network, predict changes and adapt to user behaviors without the intervention of a network operator beyond the specification of her initial intent [12]. This self-adaption is achieved through the controller by pushing out device-specific configurations or control instructions to every router/switch in the network so that the intent of the network administrator is achieved (and/or optimized). Because these systems are able to control all the devices in the network (not just at the edge – e.g., at a firewall), they can deploy fine-grained policies all throughout the network. In cases where the IBN is integrated with the network login/authentication service (e.g., Cisco's Identity Service Engine (ISE)), the system may even be able to define policies on a per-user basis.

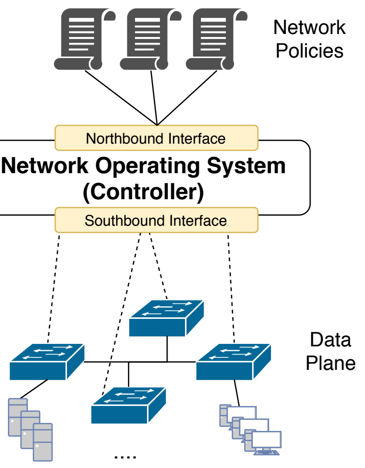


Figure 2: SDN and IBN architecture. Both are based on the concept of a controller responsible for the behavior of all nodes in the network

2.2 Programmable Networks

While a rich set of tools and applications have been developed in recent years to simplify traditionally complex and expensive tasks, computer networks have not evolved at the same pace. For the most part, network infrastructures still heavily rely on the interconnection of distributed devices that decide how to route packets in their network independently from each other based on information exchanged with neighbor nodes. Emerging workflows that use newer computing technologies such as virtualization, big data processing, or machine learning, require the network infrastructure to dynamically adapt to provide optimal results. Such specialized workflows typically lead to a substantive increase in the complexity of the configuration of network devices and a raise in the capital and operational expenditures incurred to manage the network.

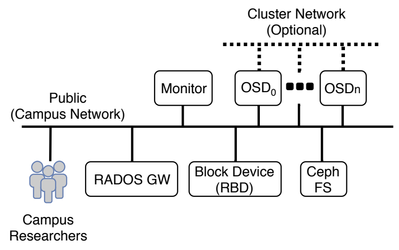
To cope with these difficulties, research environments have started to adopt newer network architectures (*e.g.*, SDN) that allow engineers to re-program the network via software. These types of networks are often called *Programmable Networks*. In a programmable network, network devices (*e.g.*, switches, routers) are dynamically reconfigured/modified by a logically-centralized entity called *controller or Network Operating System* that interacts with these devices using well-defined (standardized) protocols resembling the way traditional operating systems interact with the internals of a machine (*e.g.*, memory, NICs, I/O, etc).

Figure 2 shows an example of the overall SDN and IBN architecture where the forwarding plane is separated from the decision plane. Specifically, network devices process network traffic based on the decisions made by the controller which in turn are determined by network management software (*i.e.*, applications) that leverage the global view of the network discovered by the controller as one cohesive system. We consider that programmability of network infrastructures opens up opportunities to develop systems that can accurately influence via network policies the allowable behaviors among users and applications, and the multiple components present in today's advance research computing environments.

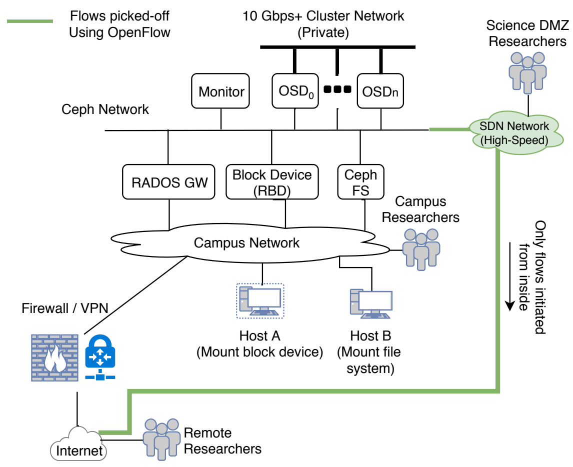
3 COMPLEX ENVIRONMENTS AND POLICIES

Conventional HPC supercomputers, as shown in Figure 1, have limited ingress and egress points which makes them easy to protect. From a network security perspective, the campus network is directly attached to the login node, allowing any on-campus user send traffic to the supercomputer. Similarly, any user with VPN access to the campus can send traffic to the supercomputer. Moreover, any user on the Internet can send traffic to the DTN node. All of these traffic avenues represent potential attack vectors – attack vectors that HPC centers have, for the most part, not been concerned about. The potential attack vectors for the types of systems that are now being deployed in HPC centers are substantially different.

Consider a Ceph Object Storage System [6]. Unlike an HPC environment where the key resources (compute nodes) only reside on private, special-purpose, hidden networks (i.e., the private infiniband data network and the private control network), the key resources in a Ceph Object Store, namely the Object Storage Daemons (OSDs) and Object Storage Monitors (OSMs), could be connected to a public general-purpose network (like the campus network).



(a)



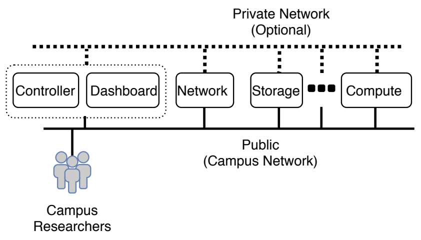
(b)

Figure 3: Basic and conceptual network architecture of Ceph

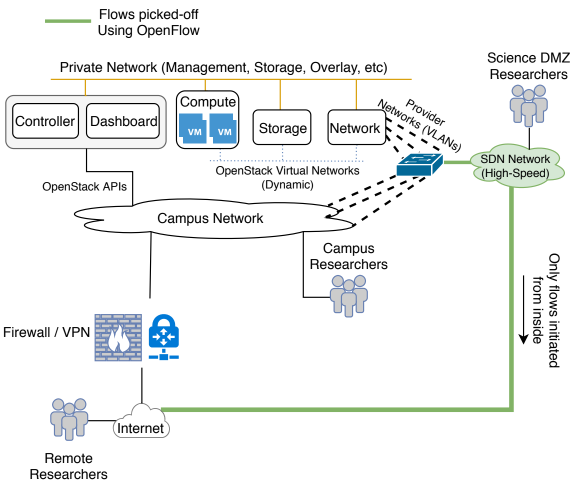
As shown in Figure 3 a, it is possible to connect all key Ceph services to the campus (public) network. In particular, the OSDs are responsible for storing objects and perform replication and recovery tasks, while the OSMs manage cluster membership and state. In addition, web services such as a RADOS gateway (RADOSGW) act as proxies that translate requests to store/retrieve data (e.g., S3 requests [1]) from user machines into Ceph-protocol messages sent to the OSDs and OSMs. Other services include a RADOS block device storage service (RBD) and a Ceph File System (CephFS), which ideally should only be reachable by machines with the privilege to mount RBD devices or CephFS file systems. The Ceph documentation recommends creating an additional high-speed special network, called the cluster network, to interconnect all the OSDs and improve replication and recovery speeds but this is not required. Because Ceph components are not on a hidden private network like HPC components, they are not automatically protected like HPC components, and thus require additional network security mechanisms to protect against attack.

To address this problem, one could deploy a Ceph system as shown in Figure 3b with additional networks (or VLANs) used to separate traffic, placing OSDs, OSMs, and RADOS servers on a distinct network. However, this still does not fully address the policy concerns. For example, it does not prohibit unauthorized hosts on the campus network from reaching (attacking) RADOS servers. Ideally, the network policy would only allow authorized CephFS clients to reach the CephFS server, and only authorized hosts to reach RDB block devices. Using programmable networks or intent-based networks, it is possible to deploy the fine-grained network policies needed to only allow reachability from authorized hosts, not the entire campus network (which may include all wireless devices on campus as well).

At the University of Kentucky, our Ceph systems are also connected to our OpenFlow-enabled SDN network (see Figure 3b) which functions as a super-configurable (i.e., programmable) Science DMZ. This allows us to dynamically create flows from Ceph to the Internet that by-pass the campus firewalls [11]. Supporting this type of service requires even more complex network policies – namely policies that allow high-speed flows to be set up between our Ceph system and certain hosts (IP addresses) in the Internet. Again, such policies can be enabled and supported through the use of programmable or intent-based mechanisms.



(a)



(b)

Figure 4: Basic and conceptual network architecture of OpenStack

As another example, consider Figure 4 which illustrates the architecture of an OpenStack system. Figure 4a illustrates a basic OpenStack network that, like Ceph, does not depend on any special-purpose, private, hidden, networks, but rather works well with general purpose public network technologies (like a campus network). Like Ceph, OpenStack recommends deploying a private management network used to provision and control the OpenStack components. In the absence of physical network separation (or virtual network separation via VLANs), any machine on campus is able to reach/attack any component of the OpenStack System, potentially including VMs deployed and managed by users. Figure 4b illustrates a potential configuration that addresses some of the issues by removing key OpenStack components from the campus (public) network and making them only accessible via the management network. It also adds the ability to create independent external "provider" networks that ensure the traffic from different VMs is isolated to the appropriate VLAN. Deploying different physical networks (or VLANs) aids with OpenStack network security, but does not necessarily support the fine-grained network security policies desired or required. Like Ceph, such policies can be enabled and supported through the use of programmable or intent-based mechanisms.

It should be noted that the physically separate networks shown in Figure 3 and Figure 4 could also be implemented using programmable networks or intent-based networks. In other words, by moving to these types of programmable solutions, it is possible to implement both fine-grained network security and security that historically has been implemented via physical or VLAN partitioning.

3.1 Example Ceph/OpenStack Policies

Given the ability to program network security policies into the network or to specify network policies using intent-specifications, one can imagine creating a variety of fine-grained, dynamic, time-dependent network security policies to protect these new types of HPC resources. Example network security polices/intents for Ceph and OpenStack systems include things such as:

- OSDs and OSMs should only be reachable by RADOS servers
- OpenStack Compute and Storage Nodes should only be reachable by the OpenStack Controller and OpenStack Network Nodes.
- The Ceph RDB server can only be accessed by the OpenStack Controller
- The Ceph CephFS server can be mounted by VMs on the OpenStack compute nodes
- User-created OpenStack VMs can be attached to the SDN network (for high-speed data transfers)
- The OpenStack VMs should only be accessible from campus or the Internet via ssh
- The OpenStack VM displays should only be accessible via VNC
- OpenStack VMs should only have access to the campus network
- OpenStack VMs should only have access to the RADOSGW
- The Ceph RADOSGW should only be accessible via the SDN network from midnight to 7am

In short, one can imagine any number of fine-grained policies that may be applied to these new and emerging HPC center resources – policies that are more complex and sophisticated and will require advanced networking capabilities such as programmable networks or intent-based networking to achieve.

4 SPECIFYING NETWORK POLICIES

Expressing network policies is a critical task for the appropriate management of a network. Even though there have been efforts to raise levels of abstraction, policies are typically expressed in terms of technical expressions or jargon that is difficult to understand, and more importantly, challenging to verify. Existing network policy mechanisms range across traditional low-level, device-specific CLI commands, to network programming languages [8, 13, 15, 22], to interactive full-fledged dashboards developed by network vendors to simplify the management of their proprietary equipment. Albeit usable, all these tools still require the specification of low-level network details such as VLAN numbers, IP addresses or port numbers. Worse, in the case of network programming languages, operators must learn non-standard programming language constructs that are hard to understand for inexperienced developers that are used to express intents via CLI commends.

In an attempt to make network policies more comprehensible (human readable) and verifiable—i.e., the intended policy is what actually was implemented in the network equipment—we have begun to develop a system that allows network operators to specify access policies in a format that is (reasonably) human readable. Moreover, given a policy statement, it can be automatically translated into network infrastructure instructions (e.g. network configurations, OpenFlow rules) that ensure the policy is correctly enforced. Our approach leverages practices found in businesses and corporations to express their policies via *Business Rule-based Management Systems (BRMS)*. Such systems not only automate the enforcement of policies on products based on different conditions and external events, but provide mechanisms to write policies in a human readable format that hides the complexities of the technicalities found on the source code that would enforce any given policy.

In the following, we describe how businesses have used BRMSs to tackle the challenges they faced when different types of policies required to be enforced using software applications. Further, we present our initial efforts to adapt these systems to implement network policies, allowing them to be specified in a human-readable fashion and then automatically converted and enforced using the OpenFlow protocol.

4.1 Adapting BRMS for Network Policies

The need to define and enforce policy based on known facts and dynamic events is not unique to network security policies. Businesses that rely on software (most of them nowadays) to provide services and manage complex business operations (*e.g.* web, payroll, accounting and sales) have experienced similar difficulties when it comes to the deployment of policies governing their processes and systems. To cope with these complexities, a class of tools called *Business Rule Management Systems (BRMS)* [16] have been developed. The key characteristic of a BRMS is the separation between the way a policy definition is specified and how the policy is actually implemented. The former is expressed in the form of rules using a BRMS-specific syntax and is typically determined by business decision makers, while the latter is done in application code written typically by trained IT developers. BRMS brings together both policy makers and policy enforcers in one place such that high-level (human readable) statements can be converted into low-level (technical) application code.

While each BRMS provides its own syntax to specify rules, they all share the same underlying structure. Specifically, a *conditional* listing conditions and constraints that must be satisfied to trigger a policy (i.e. a rule), and a *consequence* that determines the set of actions that are to be executed if such prerequisites are satisfied. We present two examples of rules in a financial institution in Figure 5:

```
Example BRMS Rule 1:
   When loan is approved
   Then send welcome e-mail

Example BRMS Rule 2:
   When amount requested < 5000 and
        credit score > 675
   Then approve loan
```

Figure 5: Example BRMS rule for a Bank

The important aspect of these rules is that unlike traditional match-action approaches found on network equipment, actions that modify the state of the system may result in other rules being triggered recursively (*e.g.* Rule 2 in our example triggers Rule 1). This feature allows rules to be "listening" constantly to new network events or actions executed by other rules and activate themselves dynamically.

We are exploring ways to achieve the benefits and features of BRMSs to simplify the definition, and automate the enforcement, of network policies in HPC environments. Our approach leverages the network-wide view and topology information provided by SDN and IBN controllers and treats this information as the set of facts that dynamically trigger BRMS rules and subsequently enforce policies.

4.2 From BRMS Network Policy to SDN Rules

To illustrate how we can adapt BRMS systems to enforce network policies for HPC environments, we present an example policy written in human readable language and show how the policy can be expressed as a rule in a BRMS and further translated into OpenFlow rules as the low-level enforcement mechanism. Recall the following OpenStack policy we listed in Section 3.1:

Policy: *The only way to reach OpenStack VM displays is via VNC from any location on campus.*

BRMS rule:

```
When node is an OpenStackVM-display
Then only-allow VNC traffic from campus network
```

Enforcement: There are multiple procedures taking place before a rule is activated. First, the controller, upon discovery of new hosts, will internally label a device as an OpenStackVM based on definition (specified elsewhere) of what an OpenStackVM-display is (*e.g.* any host in the 10.8.8.0/24 range). Upon characterization of the type of end-system, the rule will be activated and the nearest OpenFlow switch in the path to reach that node will become the enforcement point of the policy. Then, the components of the *consequence* of the rule will be translated into the low-level details of a set of OpenFlow rules. This translation may not be one-to-one. For instance, there is no "only-allow" action in OpenFlow, instead, a default rule that drops all traffic would be installed and explicit higher priority forwarding rules would be installed matching traffic from campus IPs to the port (or port-ranges) used by VNC connections (e.g. port 5900 + N, where N is the number of displays). For simplicity, we show below just the default blocking rule, and two rules letting traffic through from two different groups of campus route-able IP addresses to the physical display of the OpenStackVMs (i.e. port 5900).

OpenFlow Rules:

```
Priority: 0
Match:
  dst_ip: 10.8.8.0/24
Action: Drop
-------
Priority: 10
Match:
  src_ip: 172.16.0.0/12
  dst_ip: 10.8.8.0/24
  protocol: TCP
  dst_port: 5900
Action: Output Port 3
-------
Priority: 10
Match:
  src_ip: 128.163.0.0/16
  dst_ip: 10.8.8.0/24
  protocol: TCP
  dst_port: 5900
Action: Output Port 3
```

Even though the rules presented in the previous example are installed in an individual switch. Some policies are deployed across multiple switches with less straightforward actions than just forwarding packets through a particular port or blocking general-purpose traffic. For instance, *allowing access to Ceph RADOSGW via the SDN network*, or *letting some OpenStack VMs be part of the SDN network for high-speed data transfers* involve the calculation of optimal paths, the generation of OpenFlow rules that perform VLAN and MAC address rewriting, DNS resolution in the controller to determine IP addresses for moving targets (e.g. Google Drive), or network address translation for external transfers.

5 CONCLUSION

HPC systems are evolving with the emergence of network virtualization, programmable networks, advances in containerization and new control software. These emerging environments require more advanced, finer-grained, and dynamic network policies that make them more difficult for system administrators to manage the network security of the system. Existing approaches for policy definition that involve human intervention are well-suited for general purpose usage of the network but limited to cope with the requirements of emerging HPC systems. In this paper, we proposed an approach towards policy definition and enforcement that leverages state-of-the-art technologies such as SDN and Business Rule Management Systems to automate the deployment of network policies in campus networks. We described example security policies in HPC systems and discussed how to define them and enforce these policies by translating them into OpenFlow rules inside networks. We are looking at other technologies like Cisco's DNA, NETCONF, PBR, gRPC microservices to expand the set of events and devices that can be expressed in the policy language.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grants ACI-1541380, ACI-1541426, and ACI-1642134.

REFERENCES

[1] Amazon. 2006. Amazon Simple Storage Service. https://docs.aws.amazon.com/AmazonS3/latest/API/s3-api.pdf.

[2] Apache Mesos. 2019. What is Mesos? A distributed systems kernel. http://mesos.apache.org/.

[3] Apstra. 2019. Intent-based Networking. https://www.apstra.com/intent-based/intent-based-networking/.

[4] Big Switch Networks. 2017. Intent-based Networking with Big Cloud Fabric. https://www.bigswitch.com/videos/webinar-intent-based-networking-with-big-cloud-fabric.

[5] Sdx Central. 2019. What is Intent-based Networking. https://www.sdxcentral.com/networking/sdn/intent-based/definitions/what-is-intent-based-networking/.

[6] Ceph. 2019. The Future of Storage. https://ceph.com/.

[7] Cisco. 2019. Cisco Digital Netwokr Architecture (Cisco DNA). https://www.cisco.com/c/en/us/solutions/enterprise-networks/index.html.

[8] Douglas Comer and Adib Rastegatnia. 2018. OSDF: An Intent-based Software Defined Network Programming Framework. *2018 IEEE 43rd Conference on Local Computer Networks (LCN)* (Oct 2018). https://doi.org/10.1109/lcn.2018.8638149

[9] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. 2013. The Science DMZ: A Network Design Pattern for Data-intensive Science. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. ACM, New York, NY, USA, Article 85, 10 pages. https://doi.org/10.1145/2503210.2503245

[10] Data Center Knowledge. 2018. Intent-based Networking Data Center: Cisco vs Juniper. https://www.datacenterknowledge.com/networks/intent-based-networking-data-center-cisco-vs-juniper.

[11] J. Griffioen, K. Calvert, Z. Fei, S. Rivera, J. Chappell, M. Hayashida, C. Carpenter, Y. Song, and H. Nasir. 2017. VIP Lanes: High-Speed Custom Communication Paths for Authorized Flows. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 1–9. https://doi.org/10.1109/ICCCN.2017.8038429

[12] Arthur Selle Jacobs, Ricardo José Pfitscher, Ronaldo Alves Ferreira, and Lisandro Zambenedetti Granville. 2018. Refining Network Intents for Self-Driving Networks. In *Proceedings of the Afternoon Workshop on Self-Driving Networks (SelfDN 2018)*. ACM, New York, NY, USA, 15–21. https://doi.org/10.1145/3229584.3229590

[13] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. 2015. Kinetic: Verifiable Dynamic Network Control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 59–72.

[14] LightReading. 2019. Juniper's AppFormix Aims to Automate via Intent-Based Networking. https://www.lightreading.com/automation/junipers-appformix-aims-to-automate-via-intent-based-networking/d/d-id/736153.

[15] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. 2013. Composing Software Defined Networks. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL, 1–13.

[16] Tony Morgan. 2002. *Business rules and information systems: aligning IT with business goals*. Addison-Wesley Professional.

[17] Network Computing. 2018. Five Intent-based Networking Vendors. https://www.networkcomputing.com/networking/5-intent-based-networking-vendors.

[18] NoMachine. 2019. Fast, secure, easy way to get to your stuff. https://www.nomachine.com/.

[19] NoMachine. 2019. Managing connections by using the NX protocol. https://www.nomachine.com/FR11G02294&fn=nx%20protocol.

[20] OpenStack. 2019. Open source software for creating private and public clouds. https://www.openstack.org/.

[21] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R Wood, and Andy Hopper. 1998. Virtual network computing. *IEEE Internet Computing* 2, 1 (1998), 33–38.

[22] Robert Soulé, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer, and Nate Foster. 2014. Merlin: A Language for Provisioning Network Resources. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*. ACM, New York, NY, USA, 213–226. https://doi.org/10.1145/2674005.2674989