Elastic IP and Security Groups Implementation Using OpenFlow

Greg Stabler
School of Computing
Clemson University
Clemson, SC US
gstable@clemson.edu

Aaron Rosen
Department of Electrical and Computing Engineering
Clemson University
Clemson, SC US
arosen@clemson.edu

Sebastien Goasguen
School of Computing
Clemson University
Clemson, SC US
sebgoa@clemson.edu

Kuang-Ching Wang
Department of Electrical and Computing Engineering
Clemson University
Clemson, SC US
kwang@clemson.edu

ABSTRACT

This paper presents a reference implementation of an Elastic IP and Security Group service using the OpenFlow protocol. The implementation is the first to present integration of OpenFlow within a virtual machine provisioning engine and an API for enabling such services. In this paper the OpenNebula system is used. The Elastic IP and Security Groups services are similar to the Amazon EC2 services and present a compatible Query API implemented by OpenNebula. The core of the implementation relies on the integration of an OpenFlow controller (NOX) with the EC2 server. Flow rules can be inserted in the OpenFlow controller using the EC2 API. These rules are then used by Open vSwitch bridges on the underlying hypervisor to manage network traffic. The reference implementation presented opens the door for more advanced cloud networking services that leverage principles from software defined networking including virtual private cloud, virtual data center spanning multiple availability zones, as well as seamless migration over wide are networks.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Management

VTDC'12, June 18, 2012, Delft, The Netherlands.

Keywords

OpenFlow, software defined networking, cloud, elastic IP, security groups, firewall

1. INTRODUCTION

Cloud computing has emerged as a new paradigm for on-demand access to shared computing resources for end-users over the network. Early cloud infrastructures offered "Software As a Service" (SaaS) to end-users. They provided access to applications that were hosted on the cloud instead of local computing resources. Users accessed these applications through a web browser on their local machine. Examples of SaaS applications are Google Docs and Microsoft Office 365. Developers that wanted to write their own applications for the cloud turned to clouds that offered "Platform As a Service" (PaaS). PaaS provides the ability to deploy custom applications on the cloud without managing the underlying cloud infrastructure. These applications are developed using programming languages and tools supported by the cloud provider. Google App Engine, Windows Azure, and Heroku are examples of PaaS clouds. "Infrastructure As a Service" (IaaS) clouds are exemplified by the Amazon Web Services such as EC2 and S3 as well as Rackspace (`http://www.rackspace.com`) services. IaaS clouds allow users to provision computing resources for processing, storage, and networking. Users can control the operating systems, storage environments, and networking components of their applications deployed in the cloud without managing the underlying cloud infrastructure [14].

As enterprises adopted the cloud computing paradigm, four deployment models emerged: *private*, *community*, *public*, and *hybrid* clouds. A *private cloud* is a cloud that is used by a single organization. It may be run by the organization or managed by a third party. A *community cloud* is shared between several organizations that usually have a common interest. *Public clouds* are available to the general public and are owned and operated by an organization that sells cloud services. A cloud infrastructure that consists of two or more clouds is known as a *hybrid cloud.* The individual clouds in a hybrid infrastructure can be private, public,

or community clouds "that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability" [14].

At the IaaS layer, virtualization has been the key technology enabling on-demand, elastic resource provisioning [16]. Virtualization allows multi-tenancy of resources while providing isolation between applications. Key research thrust in on-demand provisioning of virtual machines has led to several IaaS solutions such as Nimbus [13], Eucalyptus [17] and OpenNebula [19]. Recently, OpenStack [5] has received a lot of attention as well. All of these solutions provide an EC2 interface, but few provide networking services. Cloud networking research has up till now been limited to the development of efficient overlay strategies to create networks of virtual machine instances [10], [18]. In this paper we show how OpenFlow coupled with one of these solutions can provide an implementation for networking services offered by Amazon.

CloudNaaS [9] is the closest effort related to our work. CloudNaaS is a cloud networking platform for enterprise applications that uses OpenFlow for its network management. CloudNaaS (Cloud Networking-as-a-Service) extends the self-service provisioning model for cloud services to include networking services. The CloudNaaS Network Controller is used to provision virtual network segments between network devices that meet the constraints of the requested resources. It then uses OpenFlow to join the segments to create the desired path between resources. This network segment provisioning algorithm is used to guarantee quality of service (QoS) requirements and direct traffic through middleboxes placed in the network. The resulting path is also used when considering where to place a VM in the cloud to meet latency requirements. CloudNaaS also uses OpenFlow to allow applications to use custom addressing. OpenFlow rules installed on each cloud node using a software switch translate addresses from application addresses to cloud-assigned addresses.

While CloudNaaS presents a cloud environment that uses OpenFlow to enable networking services, it is not the same as our OneCloud solution. CloudNaaS mainly uses OpenFlow to stitch together network segments between two cloud resources. It also uses OpenFlow to enable applications to use custom addressing. However, it does not use OpenFlow to enable a range of customizable network services for end-users nor does it present an API for end-users to manage the networking environment. The networking environment is controlled by the attributes of the resources being provisioned. Additionally, CloudNaaS uses OpenNebula 1.4 for its provisioning engine, which is now obsolete. OneCloud is built using the latest version of OpenNebula and its features are being integrated into the source code.

OpenFlow, a software defined networking approach for centralizing control of a network's datapath operations in one or multiple software controllers, aptly meets the needs of cloud computing users to 1) flexibly associate computing end hosts (with layer two, three, or higher addresses or contexts) with datapaths, 2) dynamically alter such associations for load balancing or failure fallback, 3) provision distinct datapath properties such as security, isolation, or quality of service, and 4) enable virtualization of the physical network into traffic "slices" and delegate their control to different admin entities. These features are useful from either a provider or a user's perspective. A provider can leverage OpenFlow to implement such user features according to its policy preferences, while users can explore a range of customization of the cloud environment to better meet their applications.

The novelty of this paper is three fold:

1. It introduces a reference implementation for one of the key networking services in IaaS providers, Elastic IP: the runtime association of a leased IP with the IP of a running virtual machine instance.

2. It introduces a reference implementation for a key security component of networking services in IaaS providers, Security Groups: the runtime firewall-like protection of a running virtual machine instance.

3. The OpenNebula provisioning engine is enhanced with a network driver that provides OpenFlow support. It builds on recently released Open vSwitch support in OpenNebula and adds full OpenFlow controller configuration. This enhancement has been committed to the OpenNebula code base and will be available in a future public release.

The rest of the paper is organized as follows. Section 2 introduces the Clemson OneCloud system and its components. Section 3 presents the Elastic IP case study enabled by OpenFlow on OneCloud. Section 4 presents the Security Group case study enabled on OneCloud by OpenFlow. Section 5 presents the envisioned roadmap for OneCloud and its potential use within GENI for cloud-based research and experimentation.

2. ONECLOUD

Clemson University OneCloud (`https://sites.google.com/site/cuonecloud/`) is an experimental cloud infrastructure based on the OpenNebula cloud framework. It is an IaaS system that enables users to provision virtual machine instances using the KVM hypervisor. OneCloud offers an EC2 front-end to its users.

OneCloud was created to research *dynamic cloud networking* infrastructures; it is available to external users upon request. Current research is focused on the dynamic networking capabilities of the OpenFlow protocol and how they can be leveraged within the cloud. Early results of developing OneCloud are presented in this paper. All hypervisors of OneCloud are physically connected on an OpenFlow network. Hypervisors use Open vSwitch [6] as the virtual machine bridge.

2.1 OpenNebula

OpenNebula is an open-source cloud computing framework "aimed at developing the industry standard solution for building and managing virtualized data centers and cloud infrastructures" [7]. The flexibility of the OpenNebula framework enables the creation and management of virtualized infrastructures that provide private, public, and hybrid IaaS clouds. It supports KVM, VMware, and Xen as the underlying hypervisors. OpenNebula also provides a centralized management interface for virtual and physical resources. OpenNebula allows seamless integration with other products and services - management tools, VM schedulers, virtual image managers, and hypervisors - through its highly extensible plug-in framework [7].

2.2 OpenFlow

OpenFlow is an API that provides cloud providers with the ability to programmatically control flows within a network. OpenFlow uses flow-based rules in order to match packets based on layer 2, 3, and/or 4 fields in addition to the switch port number on which a packet arrived. Rules apply actions to matching packets such as output on a specific port, drop the packet, rewrite packet headers, or send to controller [15]. This flexibility gives the cloud providers the opportunity to dynamically handle network changes in real time [4].

OpenFlow consists of a central control paradigm where packets not matching a flow entry are sent to a centralized controller(s) in order to make forwarding decisions. Since the OpenFlow controller can manipulate all of the switches in the network, virtual circuits can be brought up and torn down with little effort on the provider. This capability allows providers to divide the network into slices that can be managed by separate controllers, as seen in [2].

2.3 OneCloud Architecture

The OneCloud infrastructure consists of a front-end server and cluster nodes. The OneCloud front-end server is running OpenNebula 3.0 and the OpenNebula EC2 Query service, which provides an Amazon EC2 Query API compatible interface. The network within OneCloud is controlled by an OpenFlow controller running inside a virtual machine on the front-end node. Our controller application is written in Python and uses the NOX OpenFlow controller. It also exposes an XML-RPC interface for interaction with the OpenNebula EC2 API.

OneCloud cluster nodes use the Kernel-based Virtual Machine (KVM) hypervisor for virtualization. In addition, virtualized networking for each node is provided by Open vSwitch 1.2.2 with the bridge compatibility layer enabled. Open vSwitch is a software-based virtual switch that supports the OpenFlow protocol. As seen in Figure 1, each node is configured with a single Open vSwitch bridge, *br0*, that is connected to our OpenFlow controller. When a virtual machine instance is started, it is assigned a single private IP address. The instance's network interface is attached to the *br0* interface on the host node, which allows our controller to manage its network traffic.

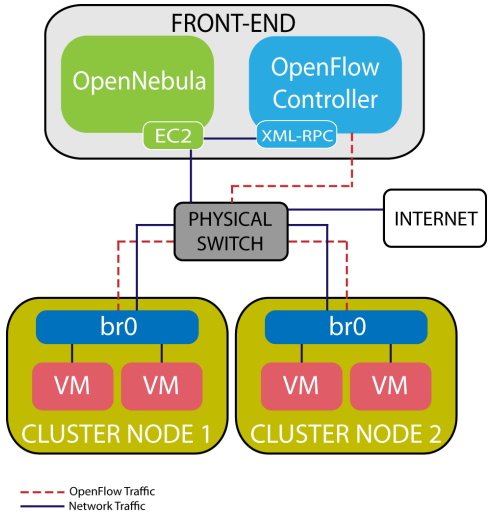

Figure 1: OneCloud Architecture

Packet Field	Replaced with...
Source IP	Elastic IP
Source MAC	Open vSwitch bridge's MAC
Destination MAC	Elastic IP Gateway's MAC
VLAN ID	Elastic IP VLAN ID

Table 1: Outgoing Packet Field Modifications

3. CASE STUDY: ELASTIC IP

Amazon Elastic IP addresses are static IP addresses that can be dynamically remapped to running instances within the cloud [1]. Each instance in Amazon's EC2 is assigned static public and private IP addresses. The public address is mapped to the private address using a 1:1 NAT mapping [1]. Once the instance is terminated, the public address is no longer valid. However, Elastic IPs are associated with a user's account and not a specific instance. Therefore, users can programmatically remap the address to a new instance in the cloud. This enables the user to mask any failures without having to wait for a DNS update to propagate through the network or a new host to be configured and brought online.

3.1 OneCloud Elastic IP

OneCloud provides users with the capabilities of EC2 Elastic IP addresses by managing the network with the OpenFlow protocol. When an instance is launched in OneCloud, its network interface is attached to the *br0* bridge on the host node. The OpenFlow controller installs rules on the bridge interface to manage the Elastic IP translations.

OneCloud users can request Elastic IP addresses using the Amazon EC2 API command *AllocateAddress*. This command instructs the OpenNebula server to reserve an Elastic IP address from the address pool and associate it with a user's account. When the user no longer needs the Elastic IP address, the API command *ReleaseAddress* instructs the OpenNebula server to return the reserved address to the address pool. Elastic IP addresses are associated with an instance using API command *AssociateAddress*. This association is removed with the command *DisassociateAddress*.

When the OpenNebula server receives the *AssociateAddress* command, it will instruct the OpenFlow controller to associate the Elastic IP address with the instance's private IP address. To do this, the OpenNebula server must provide the controller with information about the instance: private IP, MAC address, Open vSwitch bridge and port number to which the instance is attached, and the Elastic IP address. If the Elastic IP belongs to a VLAN, OpenNebula also provides the VLAN ID and gateway IP address of the Elastic IP subnet. When the OpenFlow controller receives this information, it creates a set of wildcard OpenFlow rules for incoming and outgoing IP packets to the instance.

Tables 1 and 2 show the fields in the packet headers that will be modified by the OpenFlow rules for *outgoing* and *incoming* packets respectively. In the event that packets originating from an instance's private IP are not VLAN tagged, the VLAN tag will be added or stripped away by the Open vSwitch bridge. The net effect of the installed rules is

Packet Field	Replaced with...
Destination IP	Private IP
Destination MAC	Instance's MAC
Source MAC	Open vSwitch bridge's MAC
VLAN ID	Instance's VLAN ID

Table 2: Incoming Packet Field Modifications

```
priority=32778,ip,in_port=37,nw_src=1
0.10.1.24,actions=mod_nw_src:130.127.
38.230,mod_dl_src:00:1a:a0:1d:ba:81,m
od_dl_dst:00:17:df:2b:08:00,output:3

priority=32778,ip,in_port=3,nw_dst=13
0.127.38.230,actions=mod_nw_dst:10.10
.1.24,mod_dl_src:00:1a:a0:1d:ba:81,mo
d_dl_dst:02:00:0a:0a:01:18,strip_vlan
,output:37
```

Figure 2: OpenFlow rules for Elastic IP demo

to make the Open vSwitch bridge act like a NAT router for the instances on the node.

Once these flow rules have been created, they are installed on the bridge to which the instance is attached. These rules are given a higher OpenFlow priority so that they will override any rules that were created by the basic layer 2 switch (the default behavior of the controller). The rules are installed permanently without soft or hard timeout values so they last the lifetime of the instance. When the Elastic IP is mapped to an instance, the OpenFlow controller assumes the responsibility of replying to ARP requests for the Elastic IP address.

The *DisassociateAddress* command instructs the OpenFlow controller to remove the installed flow rules and stop replying to ARP requests for the Elastic IP address.

If a user issues the *AssociateAddress* command for an address that is already mapped to a running instance, the OpenFlow controller will disassociate the address from the current instance and install flow rules to map the address to the new instance. Similarly, if the instance already has an Elastic IP address mapped to it, the existing mapping will be removed. Also, if the user terminates an instance that is mapped to an Elastic IP address, the OpenFlow controller will remove any flow entries for that address.

Because our Elastic IP implementation uses the OpenFlow controller, there is a minimal initial delay (on average less than 5 ms) in associating an Elastic IP address with a running instance. The flow rules are installed on the cluster node's bridge interface and do not have to propagate throughout the cloud. Once the rules are installed, all packet modification occurs at line-rate on the networking hardware avoiding latency due to centralized processing.

3.2 Elastic IP Example

As a test scenario, we recorded the network traffic generated by *ping* from a virtual machine instance to a physical host on the Clemson network that is not part of the OneCloud infrastructure. Our virtual machine is configured with a private IP address of 10.10.1.24.

During the test, we were assigned the Elastic IP address 130.127.38.230 from OneCloud. Figure 2 shows the OpenFlow rules installed on the cluster node to apply the Elastic IP to incoming and outgoing IP traffic. The first rule rewrites the source IP address outgoing traffic from 10.10.1.24 to 130.127.38.230 with the `mod_nw_src` action. The second rule rewrites incoming traffic's destination IP address from 130.127.38.230 to 10.10.1.24. Since our Elastic IP address is also VLAN tagged, the rules add and remove the VLAN tag as required.

Without the Elastic IP address, the virtual machine traffic is filtered through NAT on the cloud's cluster node. In our scenario, the cluster node has an IP address of 130.127.39.10. Table 3 contains the *tcpdump* trace of our *ping* session. Initially, the ICMP Echo Requests originate from 130.127.39.10, our cluster node's network interface. After mapping the Elastic IP address, the packets appear to originate from 130.127.38.230, our Elastic IP address. We then disassociate the Elastic IP address and the packets originate from our cluster node again. The trace results show a continuous flow of ICMP packets between the instance and the physical host as evidenced by the continuous sequence numbers for the same ICMP identifier value.

4. CASE STUDY: SECURITY GROUPS

OneCloud provides firewall-like services to users by implementing the Amazon EC2 Security Groups API. OneCloud's Security Group implementation uses OpenFlow rules to manage network access to virtual machines.

EC2 Security Group rules consist of a *protocol*, *from port*, *to port*, and *source*. Rules enable a specific *source* to access an instance using a certain protocol (TCP, UDP, or ICMP). For TCP and UDP traffic, the *from* and *to* ports specify a range of ports to which the rule is applied. In the case of ICMP rules, the *from port* is the ICMP type number and the *to port* is the ICMP code. The *source* can be a single IP address, a range of IP addresses, or another Security Group [8]. OneCloud currently does not support specifying another security group as the source of incoming network traffic.

Security Groups are created and removed using the API commands *CreateSecurityGroup* and *DeleteSecurityGroup* respectively. Once a security group has been created, a user can add or remove rules for *incoming packets only*. The Amazon API does not permit outgoing packet rules for EC2 Security Groups. A rule is added by issuing the command *AuthorizeSecurityGroupIngress* and removed with the command *RevokeSecurityGroupIngress*.

4.1 Launching Instances with Security Groups

When an instance is launched, the user provides a list of security groups to which the instance should belong. After the instance has been deployed to a node in the cloud, the OpenNebula server notifies the OpenFlow controller that an instance has been started. It provides the controller with the instance's private IP address, the node on which the instance is launched, and the list of security groups and their rules. The OpenFlow controller then converts the security group rules into OpenFlow rules before installing them on the switch.

Conversion of the security group rules is fairly trivial. The protocol field is translated from the EC2 API constant to the

13:14:49.387101 IP 130.127.39.10 > 130.127.39.58: ICMP echo request, id 3104, seq 4
13:14:49.387177 IP 130.127.39.58 > 130.127.39.10: ICMP echo reply, id 3104, seq 4
13:14:50.388285 IP 130.127.39.10 > 130.127.39.58: ICMP echo request, id 3104, seq 5
13:14:50.388363 IP 130.127.39.58 > 130.127.39.10: ICMP echo reply, id 3104, seq 5
13:14:51.395068 IP 130.127.38.230 > 130.127.39.58: ICMP echo request, id 3104, seq 6
13:14:51.395144 IP 130.127.39.58 > 130.127.38.230: ICMP echo reply, id 3104, seq 6
13:14:52.391292 IP 130.127.38.230 > 130.127.39.58: ICMP echo request, id 3104, seq 7
13:14:52.391368 IP 130.127.39.58 > 130.127.38.230: ICMP echo reply, id 3104, seq 7
13:14:53.392713 IP 130.127.38.230 > 130.127.39.58: ICMP echo request, id 3104, seq 8
13:14:53.392791 IP 130.127.39.58 > 130.127.38.230: ICMP echo reply, id 3104, seq 8
13:14:54.434909 IP 130.127.39.10 > 130.127.39.58: ICMP echo request, id 3104, seq 9
13:14:54.434979 IP 130.127.39.58 > 130.127.39.10: ICMP echo reply, id 3104, seq 9
13:14:55.396015 IP 130.127.39.10 > 130.127.39.58: ICMP echo request, id 3104, seq 10
13:14:55.396090 IP 130.127.39.58 > 130.127.39.10: ICMP echo reply, id 3104, seq 10

Table 3: tcpdump trace of Elastic IP network traffic

corresponding NOX controller constant. If the rule is a TCP or UDP rule and specifies a port range, then an OpenFlow rule must be created for each port in that range. The port number is used as the destination port in the rule. Additionally, the source can have one of several values: 0.0.0.0, a single IP address, or a range of IP addresses. If the source is 0.0.0.0, then a wildcard is used in the flow rule to match all source IP addresses. If the source is a single IP address, then that IP address is used in the rule. If the source is an IP range, it is either given in CIDR notation (192.168.2.0/24), or a range of IP addresses in the same subnet (i.e. 192.168.2.1- 192.168.2.20). If the source is in CIDR notation, then we can generate a single rule for the entire address block because OpenFlow supports IP subnet masks. If the source is a range of IP addresses in the same subnet, then a rule is generated for each IP address in that range.

Once the rules are converted to the OpenFlow format, the controller must install the rules on the switch. The default behavior for EC2 Security Groups is to block all new incoming traffic, allow all outgoing traffic from an instance, and allow related incoming network traffic. Therefore, the controller installs a default rule that drops all incoming packets to the instance (we will refer to this as the DROP rule). This rule is given an elevated priority over existing rules on the switch. The rules for a security group are then installed on the switch with a priority higher than the DROP rule. This ensures all packets matching the incoming rules will be allowed and all other packets will be dropped by the switch. The DROP rule and all incoming rules are installed permanently on the switch and removed only when an instance is terminated.

When an instance generates outgoing traffic, the switch installs a temporary rule allowing it to reach the network. However, the current rules for the security group may not allow the response traffic to reach the instance. Therefore, when the outbound rule is generated, the controller installs a corresponding temporary rule for the response packet. This rule is given a priority higher than the DROP rule to ensure the packet is allowed to reach the instance. Unlike the incoming rules for the security group, this rule does not have wildcard values. It matches only the flow created by the outbound traffic. Once the outbound flow of traffic stops, the idle timer on the incoming flow rule expires and it is removed from the switch.

Field	**Value**
Protocol	TCP
From Port	5001
To Port	5001
Source	10.10.1.2 - 10.10.1.5

Table 4: Example Security Group Rule

4.2 Modifying Security Group Rules

The Amazon EC2 specification [8] allows users to add or remove rules to a security group and have the changes applied to all instances in the group. OneCloud also provides this capability to its users. When a user adds a rule to a security group, the OpenNebula server notifies the OpenFlow controller of the new rule. The OpenFlow controller generates a list of instances that belong to the modified security group and installs the new rule on the switch that hosts each instance. Similarly, if the user removes a rule from a security group, the OpenFlow controller will uninstall the corresponding flow rules for each instance from their respective switches.

4.3 Security Groups Example

We present a simple scenario that demonstrates the implementation of Security Groups using OpenFlow. In our scenario, we will use a security group to restrict access to a virtual machine by the source IP address. Table 4 shows the rule for our example security group, which allows all incoming TCP traffic on port 5001 from any machine with an IP address in the range 10.10.1.2-10.10.1.5. Since this will be the only rule in our group, all other incoming traffic to the virtual machine will be dropped by the switch.

The test scenario will consist of three virtual machines in the same subnet (10.10.1.0/24) connected by a virtual switch. VM1 is launched with an IP address of 10.10.1.29 and is a member of our test security group. VM2 has an IP address of 10.10.1.3, which is within the allowed range of clients for VM1. VM3 will have an IP address of 10.10.1.23, so it will *not* be permitted to connect to VM1. To conduct the test, VM1 is running the *netcat* command in listening mode on port 5001. VM2 and VM3 will attempted to connect to VM1 using the *telnet* command.

Table 5 shows the rules generated by the OpenFlow controller for the security group. The first four rules allow TCP packets destined for VM1 on port 5001 from the IP addresses

priority=32778,tcp,dl_dst=02:00:0a:0a:01:1d,nw_src=10.10.1.4,nw_dst=10.10.1.29,tp_dst=5001 actions=output:1
priority=32778,tcp,dl_dst=02:00:0a:0a:01:1d,nw_src=10.10.1.2,nw_dst=10.10.1.29,tp_dst=5001 actions=output:1
priority=32778,tcp,dl_dst=02:00:0a:0a:01:1d,nw_src=10.10.1.3,nw_dst=10.10.1.29,tp_dst=5001 actions=output:1
priority=32778,tcp,dl_dst=02:00:0a:0a:01:1d,nw_src=10.10.1.5,nw_dst=10.10.1.29,tp_dst=5001 actions=output:1
priority=32773,ip,dl_dst=02:00:0a:0a:01:1d,nw_dst=10.10.1.29 actions=drop

Table 5: OpenFlow generated rules for example Security Group

in the specified range. The last rule drops all incoming packets to VM1. Since the last rule has a lower priority than the first four rules, the effect is to drop all packets not specifically allowed by the higher priority rules.

When VM3 attempts to connect to VM1, it sends three separate TCP SYN packets initiating a connection, as seen in the *tcpdump* output in Table 6. However, the switch is not forwarding these packets due to our security group rules. Therefore, VM1 never responds and the connection attempt times out. When VM2 attempts to connect to VM1, the switch forwards its packets because the source IP address is 10.10.1.3, which matches one of our flow rules. VM1 receives the packet and completes the TCP handshake as expected. This is seen in the last three packets of the *tcpdump* output in Table 6.

5. IMPLICATIONS ON FUTURE CLOUD EXPERIMENTATION & OPERATION

As the community continues to explore potential architecture for the future Internet, it becomes increasingly clear that one of its core strengths is the flexible insertion of computing intelligence and storage into the network fabrics. Many such examples can be observed through the demonstrations in the GENI Engineering Conference series [3]. What this implies is that networking operations such as switching, routing, and traffic engineering will always involve distributed computing and storage components in addition to the switching equipments. As a result, future application developers should no longer view the network as simply pipes plumbed together. Instead, they will benefit from viewing and leveraging the entire network as a part of their computing clouds. In the GENI context specifically, research experimenters would more easily understand the resource reservation process and, moreover, take full advantage of the new network control opportunities to advance their applications and services.

The emergence of IaaS providers has made a new set of networking services available to system administrators. These services provide a dynamically configurable network that spans multiple data centers and that needs to be interoperable between providers. OpenFlow networks offer the capabilities needed to implement such services. This paper is a foray into cloud networking services implementation using OpenFlow.

Specifically, we envision these advanced network services enabled by openflow to enhance: ***cloud creation*** (provisioning), ***cloud operation*** (management), ***cloud protection***, and ***cloud scale-up***.

Cloud creation: A cloud instance is created by a user to fulfill specific application needs. At the least, the user has a logical view of the application components and an estimation of the needed application capacity. While some users may have further preferences of how their hosts and networks should be arranged to be efficient, scalable, secure, resilient, etc., in an OpenFlow cloud environment such features can be potentially offered by the cloud provider either implicitly or explicitly through its service interface.

With OpenFlow, the logical entity for provisioning a single communication domain for an application is a *flowspace* that is associated with all the hosts, storage, and their interconnecting datapaths. In principle, an OpenFlow controller can be implemented in arbitrary ways to fulfill application specific communication needs. In practice, today's cloud users are more accustomed to reasoning within the legacy IP architecture - i.e., separate distinct IP subnet (each subnet defined over a layer 2 broadcast domain) interconnected by the IP routing protocol. In this context, it is reasonable to consider that ***a single IP subnet is sufficient to fulfill the logical need of a basic application before imposing additional requirements such as security, scalability, and resiliency***.

At this point, careful readers would already object to the omission of discussion of locality. Legacy IP subnets form the basis for routing. A single IP subnet implies a single locality (i.e., a single data center) on the global Internet for hosting the application. This limitation is no longer existent within an OpenFlow cloud. As was shown previously, ***with OpenFlow, a single IP subnet can be provisioned across multiple locations and dynamically alter its location(s) - the basis for the Elastic IP service***.

Cloud operation: Operation of a cloud must address its external interface and internal operation. Also important are security and scalability issues which will be discussed separately later. It is reasonable to expect the cloud's internal resources to not be exposed to external users. It is today's standard practice to confine all visitors to a single portal/gateway which has a public IP address and an associated domain name and place all internal hosts and storage on an internal/private subnet. The enterprise World Wide Web, the virtual private network (VPN), and the network address translation (NAT) services are good examples following this architecture. It is therefore reasonable to expect ***by default a cloud instance to be created with a private IP subnet and, if desired, one or more public IP address(es) for its service gateway(s)***. The architecture has useful implications to the enablement of portability and security for cloud instances for its ability of dynamic association between the public addresses and the internal subnets.

Cloud protection: In addition to the security provided by the above private subnet-public gateway architecture, to provide additional protection to critical services and data, it is also common practice to ***place sensitive entities in another "security-heightened" subnet***. Between this secured subnet and the default subnet a firewall server is typically placed to filter and audit all incoming and outgoing traffic (see [11] for an example).

Another form of protection is against the distributed denial of service (DDoS) attacks that are increasingly problem-

```
15:23:17.951957 IP 10.10.1.23.60137 > 10.10.1.29.5001: Flags [S], seq 1859239445
15:23:20.950738 IP 10.10.1.23.60137 > 10.10.1.29.5001: Flags [S], seq 1859239445
15:23:26.950752 IP 10.10.1.23.60137 > 10.10.1.29.5001: Flags [S], seq 1859239445
15:23:41.841364 IP 10.10.1.3.41299 > 10.10.1.29.5001: Flags [S], seq 2222899035
15:23:41.841608 IP 10.10.1.29.5001 > 10.10.1.3.41299: Flags [S.], seq 2039488407, ack 2222899036
15:23:41.871621 IP 10.10.1.3.41299 > 10.10.1.29.5001: Flags [.], ack 1
```

Table 6: *tcpdump* output of TCP traffic on the virtual switch

atic for popular service providers. In addition to the interest of detecting the onset of DDoS attempts [12], it is also useful to ***dynamically ramp up the service capability amidst an attack by launching new virtual servers and new, distinct IP subnets to cope with the short-term excessive demands***.

Cloud scale-up: In addition to the DDoS scenario described above, customer demands can increase for legitimate reasons as well. Addressing it requires either increased network throughput or increased server capacity or both. There is more than one way to increase a cloud's network and computing capacity depending on the physical and virtual organization of the infrastructure. The solution might involve utilizing more network interfaces and hosts in the data center, or it might require expanding the application across multiple data centers.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a reference implementation for Elastic IP and Security Groups using the OpenFlow protocol on Clemson's OneCloud. The implementation is the first to present integration of OpenFlow within a virtual machine provisioning engine and provide an API for networking services for end-users.

Future research in this area will include the improvement of the current Elastic IP and Security Groups implementations, and the live migration of virtual machines between data center locations.

When the mapping for an Elastic IP is removed in the current implementation, all flow rules are removed from the switch immediately. If the user is assigning a new Elastic IP to an instance, he may not want all current connections to be terminated immediately. Extensions to the current implementation will include the installation of microflow rules for existing flows that timeout once the flow has stopped generating network traffic. This would allow current connections to remain alive until the user has closed the network connection.

Amazon's EC2 Security Groups allows users to specify a security group as the source of network traffic. Our current implementation only allows users to specify IP addresses as the source of incoming traffic. Future work will investigate adding these missing features to the Security Groups implementation in OneCloud.

The OpenNebula cloud framework supports live virtual machine migration. However, the disk image must be on a shared file system that can be accessed by all of the cluster nodes. This constraint requires migrations to occur within a data center location. Our research will focus on the ability to perform live migrations between separate physical data centers. We will also investigate routing live network traffic via OpenFlow to fail-over instances located in different data centers.

7. ACKNOWLEDGMENTS

The presented work is supported in part by National Science Foundation grants CNS-0944089, OCI-0753335, OCI-1007115. The authors would also like to express their thanks to the technical supports provided by the Clemson Computing and Information Technology (CCIT), the GENI OpenFlow campus and backbone networks, and the GENI Project Office at BBN Technologies.

8. REFERENCES

[1] Feature guide: Amazon ec2 elastic ip addresses. http://aws.amazon.com/articles/1346, July 2010.

[2] Flowvisor. http://flowvisor.org, Nov 2011.

[3] Geni: Exploring networks of the future. http://www.geni.net, Nov 2011.

[4] Openflow. http://www.openflow.org, Nov 2011.

[5] Openstack. http://www.openstack.org, Nov 2011.

[6] Open vswitch. http://openvswitch.org/, February 2012.

[7] Opennebula home page. http://www.opennebula.org, January 2012.

[8] User guide for amazon elastic compute cloud. http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/, Feb 2012.

[9] T. Benson, A. Akella, A. Shaikh, and S. Sahu. Cloudnaas: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 8. ACM, 2011.

[10] A. Ganguly, A. Agrawal, P. Boykin, and R. Figueiredo. Wow: Self-organizing wide area overlay networks of virtual workstations. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 30–42. IEEE, 2006.

[11] D. Inc. Dynamic insertion of services in a multi-tenant virtual data center. http://opennetsummit.org/demonstrations.html, Oct 2011.

[12] R. Ltd. Scalable dos attack detection and mitigation. http://opennetsummit.org/demonstrations.html, Oct 2011.

[13] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 43–52. IEEE Computer Society, 2010.

[14] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus

networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[16] M. Murphy, L. Abraham, M. Fenn, and S. Goasguen. Autonomic clouds on the grid. *Journal of Grid Computing*, 8(1):1–18, 2010.

[17] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[18] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual distributed environments in a shared infrastructure. *Computer*, 38(5):63–69, 2005.

[19] B. Sotomayor, R. Montero, I. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14–22, 2009.