2011 IEEE 17th International Conference on Parallel and Distributed Systems

Network Virtualization with Cloud Virtual Switch

Hui-Min Tseng, Hui-Lan Lee, Jen-Wei Hu, Te-Lung Liu, Jee-Gong Chang, Wei-Cheng Huang
National Center for High-Performance Computing, Taiwan
{n00hmt00, gracelee, hujw, tlliu, n00cjg00, whuang}@nchc.narl.org.tw

***Abstract*—Recently, Cloud Computing is getting considerable attentions not only as the technology trend but also the new business model. Virtualization plays an extremely important role in Cloud Computing. Server virtualization has been achieved using virtualization hypervisor software such as VMware, Xen, and KVM,etc. Virtual Machines (VMs) created by hypervisors share the same physical environment with each other. Network virtualization is made by virtually connecting each virtual machine to a port on the virtual switch, which can be implemented by software or hardware. In this paper, we review both software and hardware network virtualization techniques and present the integration of open-source hypervisor with software virtual switch, which provides cloud users a more elastic and secure network environment.**

Keywords- Cloud computing, Network virtualization, Virtual Switch, Open vSwtich, OpenNebula.

I. INTRODUCTION

In recent years, Cloud computing is an emerging IT trend that provides computing resources from large scale data centers of service providers. Cloud computing has already transformed a large part of IT industry; it can be an effective strategy to reduce the IT operations and management costs [1]. According to the prediction of Gartner, worldwide Cloud services revenue will be to reach $148.8 billion in 2014 [2]. In addition to Gartner, IDC reports that spending on Cloud Service is expected to grow over five times that of traditional On-Premise IT [1].

There are several benefits of using Cloud computing that includes reducing run time and response time, minimizing the risk of deploying physical infrastructure, lowering the cost of entry, and increasing the pace of innovation [3]. For enterprises, it helps to invest in IT without having to take up big budget and provide location-independent services. For users, it provides on-demand model which we can rent machine instances with different capabilities as needed and pay at a certain per machine hour billing rate.

The key technology for emergence of Cloud computing is virtualization. Virtualization is to give each program the illusion that it has its own hardware. The virtual machine is depended on this technology to execute application programs like a real machine. Each virtual machine resides on a hypervisor or Virtual Machine Monitor (VMM). CPU virtualization has evolved rapidly over the last decade, but virtual networks have lagged behind [4]. The reason is that Grid and Cloud computing consider the service interaction and do not take the network infrastructure. In most cases, those frameworks focus on processing of batch jobs with no timing and location constraints, which can be executed in huge remote data centers [5].

For virtual machines reside in host machines, they share same physical resources including network connection. Network virtualization is made by virtually connecting each virtual machine to a port on the virtual switch, which can be implemented by software or hardware. Currently, most of virtualization environments in a cloud IDC are using either software-based virtual switches or hardware-based virtual switch. Owing to the emergence of cloud computing service, the number of virtual switches begins to dramatically expand. This leads to the management complexity, security issues and even performance degradation. In early stage of virtualization deployment, the roles of server administrator and network administrator become confused, as the software-based virtual switch is typically managed by the server administrator rather than network administrator. To obtain management flexibility and higher performance, the software-based virtual switch is moving into hardware on the physical switch using VEPA[6] and VN-Tag[7] technology. Moreover, hardware-based virtual network switch technology VEPA and VN-Tag would offload switching from the server to access layer switch. Hence, management of physical and virtual switching can be unified under a single management point by the network administrator.

The rest of this paper is organized as follows. In Section II, we introduce the software-based virtual switch. In section III, we present two draft protocols, IEEE 802.1Qbg and IEEE 802.1Qbh, which implement hardware-based virtual switch with VEPA and VN-Tag techniques. Integration of open-source hypervisor with virtual switch technology is exhibited in Section IV. Finally, we give a short conclusion in Section V.

II. SOFTWARE-BASED VIRTAUL SWITCH

The 80x86 hypervisors introduced in the late 1990s implemented or used the existed network bridge to provide virtual machines connectivity [8]. This network component is called Virtual Ethernet Bridge (VEB) [9] or virtual switch (vSwitch), such as the bridge module in common Linux distributions. Each virtual machine in VMM has least one virtual network interface cards (vNICs) that are shared physical network interface cards (pNICs) on the physical host through vSwitch, as illustrated in Fig. 2. These virtual switches are lacking of management support and traffic visibility. For example, in order to separating various VM users, we may assign different subnets which have its distinct range of private IP address. Although this can be work properly in network connectivity, all users are still in the

1521-9097/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPADS.2011.159

same broadcast domain. If there are intentional or unwilled users modifying their own VM IP address to other IP ranges, they can access another user's VMs. This causes the difficulties of network management and security in Cloud environment. At low complexity scale, it may be a minor issue. From [8], we expect that the number of VMs per server has increased greatly. Therefore, network and server administrator must separate packets from different VM users effectively to avoid such security problem. Another issue is traffic visibility. As shown in Figure 1, for two or more VMs that reside in the same physical machine, their packets will stay inside the host and will be not sent to outside switch. If there is an malicious programs attack among these VMs, it will not be detected by current security devices such as IDS or IPS.

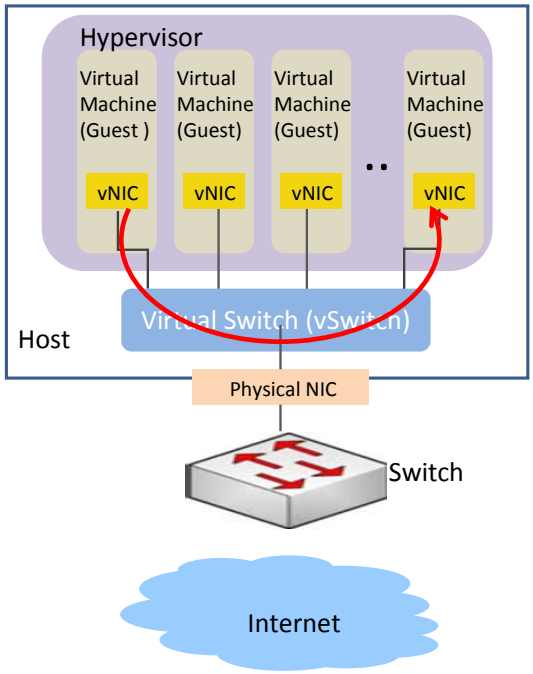

Figure 1. Virtual switch and physical switch

The vSwitch originally lack of advanced networking features such as VLAN, port mirror and port channel etc. With the matured growth of the cloud computing, various advanced features are developed for supporting virtualization. Nowadays, some hypervisor vSwitch vendors provide VLAN assignment as well as traffic monitor and can also configure parameters for ACL and QoS. A number of commercial vendors dedicated to developing hypervisor vSwitch, for example, VMware and Hyper-V. VMware's solution is a sophisticated and proprietary software switching platform. However, Open vSwitch, an open source virtual switch, may be superior to production software in quality. Open vSwitch not only can be fully integrated into XenServer as a default vSwitch but also can be used in most of Linux-based hypervisor including KVM and Xen.

Open vSwitch is a multilayer virtual switch licensed under the open source Apache 2 license [10]. It is written in platform-independent C and is easily ported to other environments, include hardware switch chipset. Open vSwitch implements 802.1q VLAN, that can really isolate different VM users to keep inter-VM security. In addition to VLAN tagging, it also supports NetFlow, RSPAN for network visibility and ACLs for network access management.

There are many utilities or platforms for supporting network virtualization, which often use common existed Linux bridge to be its network element. As mentioned earlier, the bridge module is less functionality than Open vSwitch. Thus, Open vSwitch also has a module which is compatible with Linux bridge for some useful tools or platforms, such as libvirt, and OpenNebula. The integration of Open vSwitch and OpenNebula will be presented in Section IV.

III. Hardware-Based Virtaul Switch

Although there are advanced software-based virtual switch solutions such as Open vSwitch that supports user group separation and traffic visibility, software-based technique still suffer from some drawbacks. First, software virtual switches are installed inside physical host, which will consume CPU and memory usage. Another issue is the administration of network configuration. Traditionally, network configuration is designed and managed by network administrators. When we install software-based switches in cloud servers, configurations of network is now divided into two parts: physical network devices are managed by network administrators, while software virtual switches are configured by server administrators. Possible inconsistence of network and server configurations may cause errors and is very hard to troubleshooting and maintenance.

Therefore, hardware-based virtual switch solution emerges for better resource utilization and configuration consistence. In this section, we present two ongoing IEEE draft standards that adopt different techniques for implement hardware-based virtual switch.

A. Virtual Ethernet Port Aggregator (VEPA)

Virtual Ethernet Port Aggregator (VEPA) standardization is lead by HP and there are other companies also working on it including Extreme, IBM, Brocade and Juniper etc. VEPA is being positioned as an emerging technology, as part of IEEE 802.1Qbg Edge Virtual Bridge (EVB)[12] standard. The main goal of VEPA is to allow traffic of VM to exit and re-enter the same server physical port to enable switching among VMs.

As depicted in Figure 2, minor VEPA software update is required for host servers in order to force packets to be transmitted to external switches. In a traditional access layer, switches are able to forward packets out to every ports or a specific destination port except the one it was originally received from. Without external VEPA enabled switch, the VM reside on a physical server will not communicate with another VM in the same server because the source MAC address and destination MAC address were learned on the same interface. And finally they will be silently dropped by external switch.

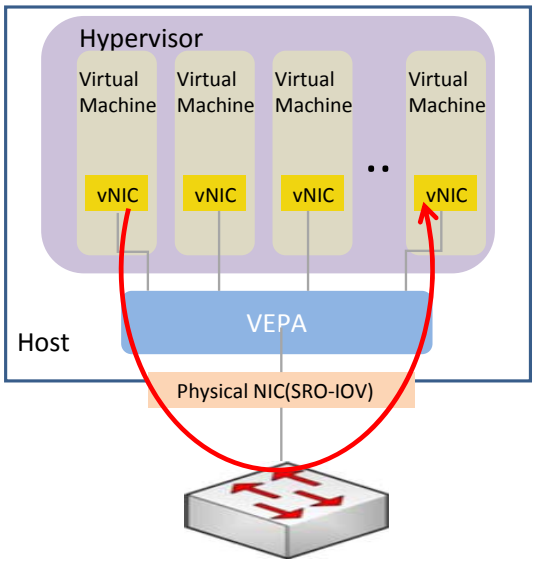

Figure 2. Virtual Ethernet Port Aggregator (VEPA)

VEPA takes advantage of "hairpin turn/reflective relay" forwarding mode on the physical network switch which allows traffic to "hairpin" back out the same port it just received it from, as illustrated in Figure 2. Such hairpin mode also requires firmware update to existing switches. After the packets received by the host, VEPA will decide the destined VM according to its MAC/VLAN-based table. Therefore, we can exploit existing hardware investment by minor software/firmware update and the network configuration will maintained by external switches. However, VEPA still consumes server resources in order to perform forwarding table lookup.

B. VN-Tag

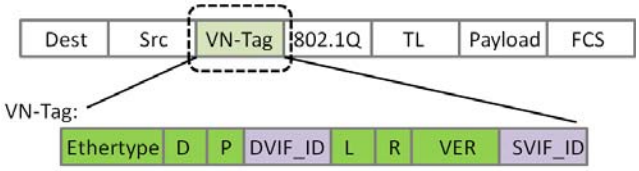

Figure 3. VN-Tag Format

Another network standard is to enable the switching function between Virtual Machine to be performed totally by an external physical network device. The VN-Tag is proposed by Cisco and it has been adopted by the IEEE as the basis for 802.1Qbh 'Bridge Port Extension'. At present, both IEEE 802.1Qbh and IEEE 802.1Qbg are in the draft stage. Compared with 802.1Qbg, 802.1Qbh is still very early in the standard process and it is attributable mainly that 802.1Qbh will modify the Ethernet frame format. 802.1Qbh must insert a VN-Tag between Layer2 and 802.1Q header in order to indicate what path the frame should travel on its way to another Virtual Machine. The VN-Tag format is depicted in Figure 3.

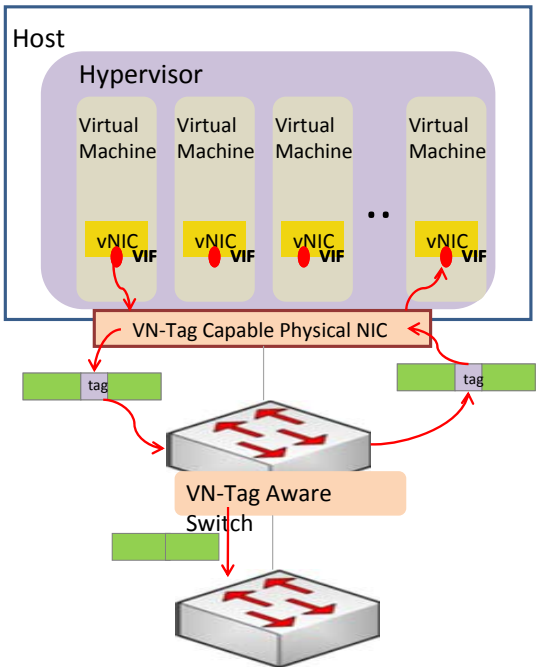

Figure 4. VN-Tag Operation

The VN-Tag contains two essential fields: the source virtual interface identifier (SVIF_ID) of the sending host and the destination virtual interface identifier (DVIF_ID) of the receiving host. A VIF [7] is any individual interface that should be treated independently on the virtual machine but shares a physical port with other interfaces. It has the advantage of allowing for individual configuration and management of each virtual interface in Virtual Machine as if it was a physical port.

Undoubtedly, the VN-Tag capable NIC and the VN-Tag aware switch are the necessary hardware components to ensure this mechanism is running well. The VN-Tag capable NICs insert tag at layer2 of the data frame that identify a network packet as part of a specified VM. The VN-Tag aware switches must be capable of stripping the tag information from the frame header at the stage of forwarding to normal switch. Also, the switch must be capable of converting a stripped or untagged frame, if the packet is to be transmitted to a VM destination via a VIF port. The detailed operation processing is as show in Figure4.

With VN-Tag technique, the switching will be totally performed by external VN-Tag aware switches. No forwarding decision table will be left in host servers. However, because the Ethernet frame format is modified, we have to invest new host NICs and external switches in order to recognize VN-Tag.

IV. Integration of Virtual SWitch with Hypervisor

In the Open vSwitch introduction in Section II, we mentioned that it supports Linux-based hypervisor and compatible with Linux bridge module for collaborating with toolkits like libvirt, and OpenNebula. In this Section, we will demonstrate the integration of Open vSwitch with OpenNebula. When a user request and deploy a new VM, a

virtual interface is automatically generated for the VM attached with users's own VLAN ID. Packets of various users are separated since each user has their own VLAN ID. Hence, network virtualization can be implemented elastically and securely.

A. *Open vSwitch Testing Environment*

In order to resolve the network separation issue for different VM users, we choose Open vSwitch (OVS) to replace Linux Kernel Bridge Module in the host machines. Compared with Linux Bridge Module which conforms to IEEE 802.1d for basic bridge operation, Open vSwitch supports IEEE 802.1Q that enables virtual LAN function. We can attach VLAN ID to Linux virtual interfaces in order to indicate different user group. Each user will have its own LAN environment separated from other users.

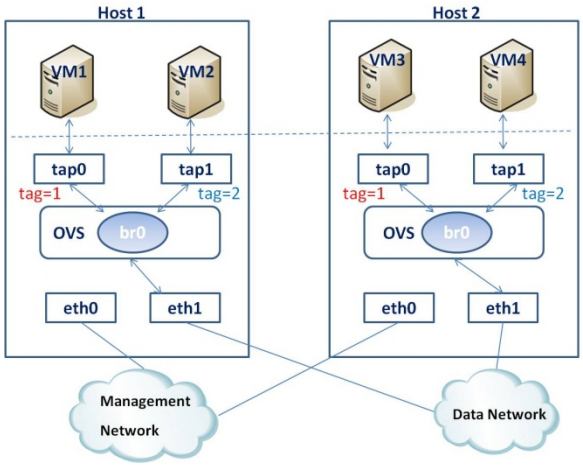

Figure 5. Open vSwitch Testing Enviromnet

Figure 5 illustrates our testing environment. There are two physical servers Host1 and Host2, each contains two Virtual Machines. VM1 and VM2 are executed on Host1 while VM3 and VM3 run over Host2. In order to exhibit VLAN partitioning, we assume that VM1 and VM3 belong to a user assigned with VLAN ID 1 through tap0 virtual interface; VM2 and VM4 belong to a user assigned with VLAN ID 2 through tap1 virtual interface. There are two external networks attached with both hosts: Data Network is used for VMs to transmit data packets; Management Network is used for out-of-band management and packet mirroring. On both hosts, we have to firstly install an OVS bridge interface (br0) for performing virtual switch function and attach it with physical Data Network interface eth1. Then we attach virtual interfaces of each VM (tap0 and tap1) to the OVS bridge interface (br0) with proper VLAN ID as follows:

```
#ovs-vsctl add-br br0
#ovs-vsctl add-port br0 eth1
#ovs-vsctl add-port br0 tap0 tag=1
#ovs-vsctl add-port br0 tap0 tag=2
```

Figure 6. Open vSwitch Control Commands

After we complete the commands shown above, we could assign VM1~VM4 within the same IP range. Basically, VM1 and VM3 can ping each other and VM2 and VM4 can ping each other successfully. We can further testify VLAN operation that VM1 cannot access VM2 and VM3 cannot access VM4 even they exist on the same host machine. To facilitate with existing virtual machine hypervisors such as KVM, we load the Linux bridge compatible module. Thus, we can control Open vSwitch with Linux bridge command tool brctl, also we could manage our VM network interface by libvirt toolkit.

B. *Network configration with libvirt*

Libvirt[12] is an management interface for virtualization environment. It supports various kinds of hypervisors, including KVM, XEN, OEMU, VMWare, etc. We can control libvirt functions with virsh command-line interface or virt-manager GUI as shown in Figure 6. The functions of libvirt contain:

1. VM Management: Controls operations of VM such as start/stop, suspend/resume, save/restore, and migration.
2. Remote Machine Managemet: Supports management functions to VMs on remote hosts.
3. Storage Management: provides virtualized shared storage pool.
4. Network Interface Management: Manages networks with NAT or route-based configureation.

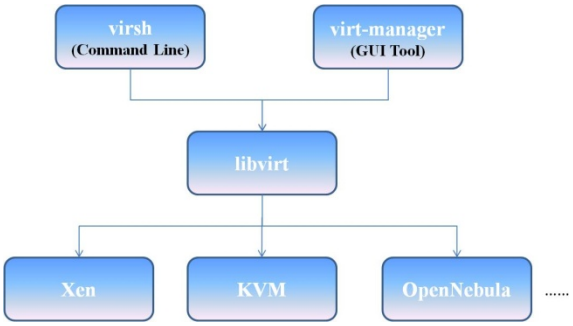

Figure 7. Architecute of libvirt

Because libvirt could manage several hypervisors easily, we allocate virtual machines with KVM hypervisor in Ubuntu Linux distribution using libvirt. As depicted in Figure 8, we execute virt-install command-line to create a VM called VM32, with parameters describing that there are 256MB system memory, OS is Ubuntu Linux that installed from image file at local disk, image file sizes 10GB, network settings choose default configuration, and hypervisor is KVM.

```
#virt-install --connect qemu:///system --name=vm32 --ram=256
--os-type=linux--os-variant=ubuntujaunty
--cdrom=/home/nchc/ubuntu-10.04.2-desktop-i386.iso
--file=/home/nchc/newvm/vm32.qcow2 --file-size=10
--network network-default --kvm --accelerate --vnc --noautoconsole
```

Figure 8. libvirt Command Line for VM Creation

After the new VM is created, there is an XML description file generated automatically at /etc/libvirt/qemu directory. When we start the VM at first time, it will load Ubuntu installation process from local disk image and we can setup the OS of this VM following the installation procedure.

```
<interface type=`network`>
  <mac address=`52:54:00:ee:bc:e4`/>
  <source network=`default`/>
  <target dev=`nchctap0`/>
  <model type=`virtio`/>
  <address type=`pci` domain=`0x000` bus=`0x00`
    slot=`0x03` function=`0x0`/>
</interface>
```

Figure 9. Modifying target Attribute in XML Description File

By default, VMs created by libvirt will automatically attach to virtual network interfaces named vnet*x*,.where *x* is an incremental serial number. In consequence, it is inconvenient to follow the serial number in order to determine the virtual network interface name of the new VM, which is the mandatory parameter for attaching OVS bridge module with proper VLAN ID in Figure 6. To resolve this issue, we could modify the XML description file of the VM with "target" attribute. As shown if Figure 9, we insert the target attribute that force the network interface name as nchctap0. After the VM created, we can use this name as the parameter to Open vSwitch commands.

Although manually modification to XML files could make it clear to indicate the virtual network interface, it will cause great management loading when numbers of VM increases dramatically. Hence, we need an automatic solution for supporting such large scale environment. Normally, we use toolkits like OpenNebula to manage VMs across numbers of physical hosts. In the following subsection, we will present how virtual network configuration will be automatically executed by integrating with OpenNubula.

C. *OpenNebula Integration*

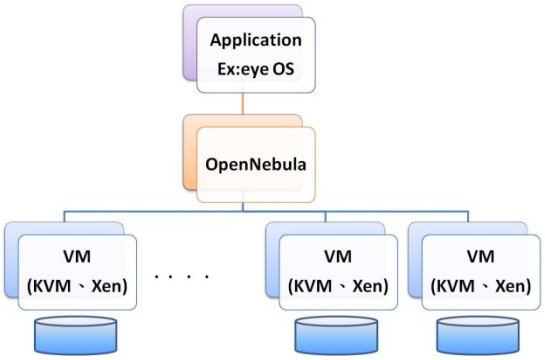

Figure 10. OpenNebula Architecture

OpenNebula (ONE)[13] is an open-source toolkit for cloud computing. It could dynamically deploy, manage, and monitor virtual machines over various physical host machines and is suitable for public, private, and hybrid cloud environment. As illustrated in Figure 10, there is an virtualization tier over existing hypervisors like Xen and KVM in the OpenNebula architecture. Such OpenNebula tier also leaves API for communicating with upper applications like eye OS.

We can create and deploy VMs through OpenNebula tools with pre-defined VM description files (*.one) as shown in Figure 11. In this example, we define the VM with its name vm32, one CPU, 256MB system memory, x86_64 OS kernel booting from HD, Ubuntu installation image, DHCP IP allocation, and etc. OpenNebula will create VMs according to the parameters descried in *.one files.

```
#-------------------------------------------------------------
# VM Definition Example
#-------------------------------------------------------------

NAME = "vm32"

CPU = 1
MEMORY = 256

# --- kernel & boot device ---
OS = [
  arch = "x86_64",
  boot = "hd" ]

# --- 1 disk ---
DISK = [
  image = "Ubuntu 10.10 Desktop x86_64",
  target = "vda",
  bus = "virtio",
  DRIVER = "qcow2" ]

# --- 1 NIC ---
NIC = [
  NETWORK = "Private IP",
  MODEL= "virtio",
  MAC = "52:54:00:ee:bc:e4",
  TARGET  = "nchctap0"  ]
```

Figure 11. Sample VM Configuration File in OpenNebula

Similarly, network configuration section in *.one files includes TARGET attribute for indicating virtual network interface name. In the previous example, we set TARGET = "nchctap0" and reload the *.one file. The new VM will attach with virtual network interface named nchctap0. In our implementation, we have upper eye OS application that interfaces with cloud users. When a cloud user request a VM dynamically, eye OS will create such *.one file that indicate sa specific named virtual network interface in TARGET attribute. Note that we still need VLAN ID information for the interface, therefore, the eye OS application will encode user's VLAN ID inside the MAC attribute in *.one file.

After VM is created and deployed, we still have to manually execute Open vSwitch commands in Figure 6 for VLAN assignment. Fortunately, OpenNabula toolkit provides Hook manager that enables administrators to customize their own script programs under several circumstances. We exploit Hook manage to execute our Open vSwitch script after a VM is started. The flow chart is illustrated in Figure 12 and described as follows:

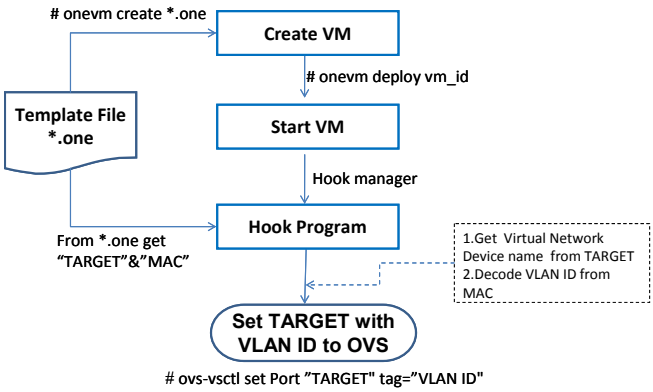

Figure 12. Flow Chart for Integration of OpenNebula and Open vSwitch After a VM is Created and Deployed.

- When a user requests a VM in eye OS UI, their virtual network interface name is generated as TARGET attribute and their VLAN ID is encoded in MAC attribute defined in Template *.one file.
- OpenNebula create and start the VM according to the parameters defined in *.one file and the hook script program is called by Hook manager.
- The Hook program read the *.one file. The virtual network interface name is the TARGET attribute while VLAN ID is decoded from MAC attribute.
- The Hook program then execute Open vSwitch command #ovs-vsctl set Port "TARGET" tag="VLAN ID" to furnish VLAN configuration.

With the integration described above, users could specify their VM requests in eye OS UI. Their virtual network interface and VLAN ID will be generated and defined in *.one file. After the VM is deployed by OpenNebula, the hook program will be automatically executed to operate Open vSwitch settings. Therefore, each user will have their VMs created within the same secured VLAN environment and no manual configuration is required.

V. Conclution

Although network virtualization draws little attention currently, it plays an important role in cloud environment. With the explosion of cloud services, the scale of cloud IDC will grow rapidly, which makes network virtualization a key emerging technology. In this paper, we review both software-based and hardware-based cloud virtual switch techniques. Open vSwitch, a promising software-based solution, resolves the problems such as network separation and traffic visibility. It can be integrated with open-source hypervisors and managed by OpenNebula toolkit. Within the demonstrated integrated environment, the cloud users can be assigned VMs with elastic and secure network configurations. It should be also noted that there are two draft hardware virtual switch standards IEEE 802.1Qbg and 802.1Qbh. There will be various mature hardware switch products that support virtual switching after the standards finalized. Cloud IDC providers should carefully keep eyes on these developing technologies and design their own expandable service architecture in the future.

References

[1] Ranjith Ramakrishnan, "What is cloud computing?," CUMULUX, http://www.cumulux.com/Cloud Computing Primer.pdf.

[2] Gartner Newsroom, http://www.gartner.com/it/page.jsp?id=1389313.

[3] R. Barga, J. Bernabeu-Auban, D. Gannon, and C. Poulain , "Cloud Computing Architecture and Application Programming," ACM SIGACT News vol. 40 issue 2, pp. 94-95, June 2009

[4] J. Pettit, J. Gross, B. Pfaff, and M Casado, "Virtual Switching in an Era of Advanced Edges, "2nd Workshop on Data Center – Converged and Virtual Ethernet Switching (DC-CAVES), ITC 22, September 2010.

[5] Karsten Oberle, Marcus Kessler, Manuel Stein, Thomas Voith, Dominik Lamp, Sören Berger, "Network Virtualization: The missing piece," ICIN conference, October 2009.

[6] P. Congdon. Virtual Ethernet Port Aggregator Standards Body Discussion. http://www.ieee802.org/1/files/public/docs2008/new-congdon-vepa-1108-v01.pdf, Nov. 2008.

[7] J. Pelissier. VNTag 101. http://www.ieee802.org/1/files/public/docs2009/new-pelissier-vntag-seminar-0508.pdf, May 2008.

[8] B. Pfaff, J. Pettit, K.A.T. Koponen, M. Casado, S. Shenker, "Extending networking into the virtualization layer," Proceedings of the ACM SIGCOMM HotNets, October 2009

[9] J. Onisick, "Access Layer Network Virtualization: VN-Tag and VEPA, " http://wikibon.org/wiki/v/Edge_Virtual_Bridging.

[10] Open vSwitch, http://www.openvswitch.org/

[11] IEEE 802.1Qbg - Edge Virtual Bridging, http://www.ieee802.org/1/pages/802.1bg.html

[12] libvirt, http://libvirt.org/.

[13] OpenNebula, http://opennebula.org/.