Zeppelin - A Third Generation Data Center Network Virtualization Technology based on SDN and MPLS

James Kempf, Ying Zhang, Ramesh Mishra, Neda Beheshti
Ericsson Research

***Abstract*—Just like computation and storage, networks in data centers require virtualization in order to provide isolation between multiple co-existing tenants. Existing data center network virtualization approaches can be roughly divided into two generations: a first generation approach using simple VLANs and MAC addresses in various ways to achieve isolation and a second generation approach using IP overlay networks. These approaches suffer drawbacks. VLAN and MAC based approaches are difficult to manage and tie VM networking directly into the physical infrastructure, reducing flexibility in VM placement and movement. IP overlay networks typically have an relatively low scalability limit in the number of tenant VMs that can participate in a virtual network and problems are difficult to debug. In addition, none of the approaches meshes easily with existing provider wide area VPN technology, which uses MPLS. In this paper, we propose a third generation approach: multiple layers of tags to achieve isolation and designate routes through the data center network. The tagging protocol can be either carrier Ethernet or MPLS, both of which support multiple layers of tags. We illustrate this approach with a scheme called Zeppelin: packet tagging using MPLS with a centralized SDN control plane implementing Openflow control of the data center switches.**

I. Introduction

Cloud computing has experienced explosive growth, thanks to the proliferation of virtualization techniques, commodity devices and rich Internet connectivity. Cloud customers outsource their computation and storage to public providers and pay for the service usage on demand with the "pay-as-you-go" charging model. This model offers unprecedented advantages in terms of cost and reliability, compared to the traditional computing model that uses dedicated, in-house infrastructure. Today, a number of companies have provided public cloud computing services, such as Amazon's Elastic Compute Cloud (EC2) [1], Google's Google App Engine [2], Microsofts Azure Service Platform [3], Rackspaces Mosso [4], and GoGrid [5].

To maximize economic benefits and resource utilization, the cloud provider usually simultaneously initiates multiple virtual machines to execute on the same physical server. Moreover, the cloud providers use "multi-tenancy", where virtual machines from multiple customers, or tenants, can share the same machine. Most cloud providers only use host based virtualization technologies to realize separation and performance isolation between VMs on the end host machine level. In the network, the cloud provider deploys and manages a set of physical routers and links that carry traffic for all customers indistinguishably. The Service Level Agreement (SLA) defined in todays cloud computing are mainly centered around the computation and storage resources. An application's performance is still unpredictable due to the lack of guarantee in the network layer. Therefore, many companies are still reluctant to move their services or enterprise applications to the cloud, due to reliability, performance, security and privacy concerns.

Thus, it is critical for cloud operators to deploy efficient traffic isolation techniques, for isolating performance among tenants, minimizing disruption, as well as preventing malicious DoS attacks. In this work, we aim at providing a scalable and effective solution for traffic isolation in the cloud platform. We categorize the existing work on cloud network virtualization into two generations: a first generation approach using simple VLANs and MAC addresses to partition the traffic according to the specific tenant. The VLAN and MAC address based approach has poor manageability, *i.e.,* is difficult to configure and limited in the number of tenants it supports. In this approach, the mapping of virtual networks to the physical network is static, leaving little choice for dynamic VM placement and movement. To overcome these limitations, several new techniques using IP overlay networks have been proposed [6], [7], [8], considered as the second generation. The approach using IP overlay networks suffers from a different scalability problem. Typically it can only support hundreds of VMs per virtual network. In addition, mapping between the on-the-wire packets and the tenant's VMs emitting the traffic is not transparent, complicating debugging problems.

Our scheme, called Zeppelin, builds upon MPLS [9], which is widely used in provider provisioned Virtual Private Networks and commonly supported in various switches. We propose an OpenFlow based controller application that oversees a set of MPLS based forwarding elements. We rely on the OpenFlow capable virtual switch on each physical server to encapsulate packets with a tenant-specific label, and a label specifying the last hop link of the destination VM. The OpenFlow controller maintains the mapping between the destination VM's address and the destination MPLS link label. To improve scalability, we use hierarchical labels, *i.e.,* separating the tenant labels from the location labels. Zeppelin is an example of a third generation data center virtualization scheme, based on multiple layers of packet tagging instead of a single layer as with simple VLANs.

We summarize the main contribution of this paper below. We propose a new cloud network virtualization architecture built on top of the MPLS and OpenFlow technologies. We

illustrate its usefulness on various use cases, such as VM creation and VM movement. We examine the correctness of our design by implementing it on the Mininet emulator, and perform an analysis to verify scalability.

This paper is organized as follows. We discuss the existing work in Section II. The design of Zeppelin is presented in Section III and the implementation is described in Section IV. We perform analysis on the scalability in Section V and conclude in Section VI.

II. Related Work

The first generation of data center network virtualization uses simple VLANs or MAC addresses in various ways. In a data center, VLANs are used to partition the traffic according to the specific tenant that the VM belongs to. Such groups are identified by their unique VLAN ID. When the VLAN ID is restricted to 802.1q [10], the maximum size of the VLAN ID is 12 bits, limiting the number of tenants the virtualized data center network can support. In addition, if a server supports VMs from more than one tenant, the link between the server NIC and the first hop switch must be trunked. Managing VLAN trunks across multiple server to switch links is complicated and error-prone. Other simple MAC approaches, such as MAC address filtering or simple MAC in MAC encapsulation, suffer from similar scalability and managability constraints, especially when the traditional Ethernet distributed control plane is used.

To overcome these issues, IP encapsulation approaches were developed. The basis of these approaches is a mesh of IP tunnels connecting the virtual switches on servers supporting the same tenant. The tenant Layer 2 traffic is isolated inside the IP tunnels. VXLAN (Virtual eXtensible LAN) technology [7], is an example of such an approach. The tunneling is realized by encapsulating Ethernet frames with four additional protocol headers: the outer Ethernet, outer IP, outer UDP and VXLAN headers. The VXLAN tunnel is set up between two end points (VTEP) whose IP addresses are used as the outer IP addresses. The VXLAN header is used to encode a 24 bit VXLAN Network Identifier (VNI) or segment ID, which can support up to 16 million separate virtual networks. The VTEPs, which are implemented in the virtual switch of hypervisors or the access switches, encapsulate packets with the VM's associated VNI. The realization of traffic isolation is implemented via an IP multicast scheme. Each VNI is mapped into a multicast group. The Internet Group Management Protocol (IGMP) messages are sent from the VTEP to the upstream switches to join and leave the group. The messages are only sent to members within the same multicast group, eliminating unnecessary unicast flooding. Another example of IP overlay virtualization is NVP [8] which uses Generic Route Encapsulation (GRE) tunnels [6] (among others). IP overlay approaches typically use centralized control planes where the IP addresses of the tenant overlays are tracked. This approach is often called software defined networking (SDN) [11].

The IP overlay approach achieves a high degree of scalability in the number of tenant virtual networks that can be supported at the expense of the number of VMs per virtual network. While the topology of the tunnels can be optimized for a particular application, without loss of generality, it is suggested that a full mesh is used, *i.e.,* setting up a tunnel between any pair of hypervisors in the tenant's virtual network. As the number of VMs increases, the number of virtual switches that must be updated when a tenant VM joins or leaves the network becomes unwieldy. Equally, the lack of tools for debugging problems in overlay networks makes network management more difficult than with solutions that use the underlying hardware support for traffic isolation.

A third generation of data center virtualization technology utilizes modern tagging based approaches, such as SPBM (802.1aq [12]) or MPLS [9]. This architecture uses multiple layers of packet tags to designate tenant virtual networks and routing through the data center physical network fabric. While VLAN-based virtualization approaches also use tags, only one tag is available. With multiple layers of tags, scalability in the number of networks can be achieved. Because only the networking elements involved in routing packets between two VMs need to be touched to set up routing, scalability in the number of VMs can be achieved. A mixture of centralized and distributed control planes is used, with smaller data centers favoring a centralized control plane and larger data centers favoring a distributed control plane. Recently, there have been efforts on building virtual overlay networks using SDN [13]. In Zeppelin, we use MPLS for packet tagging and a centralized control plane with OpenFlow [14] as the control protocol for setting up flow state on the switches. For an example of a distributed data center fabric using SPBM, see Allan, et. al. [15].

Flowvisor [16] represents another approach to cloud network virtualization based on OpenFlow. With Flowvisor, the data center OpenFlow controller uses all fields in the header to partition up the network, not just VLAN tags or MPLS labels. The header fields of the tenants are rewritten to map their flows into subsets of the header to isolate the tenant's traffic. In principle, this can allow two tenants to share the same VLAN tag if their address spaces overlap but they do not use all the addresses, for example, reducing the possibility that the VLAN tag space will run out. While Flowvisor represents an innovate approach to using OpenFlow for data center virtualization, there are some issues. Providing tenants with visibility into what fields are being remapped is required, and debugging problems becomes more complicated because the header fields on the wire don't represent a simple map of the tenant's network address space. In contrast, debugging the multi-tagging virtualized networks is much simpler, because there is a straightforward mapping between the tenant and the tags.

III. Design

The physical infrastructure of data center networks is typically designed around a rack of 20-40 servers connected into a top of rack switch (TORS). The TORS aggregates traffic from the servers in the rack and feeds it into the data

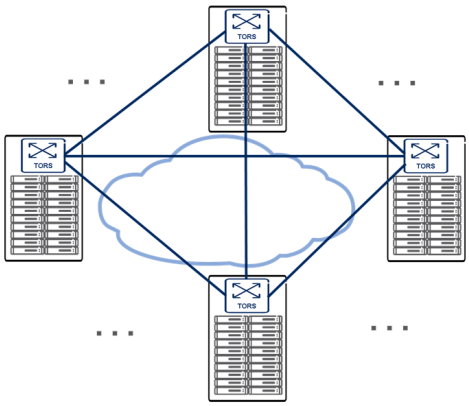

Fig. 1. Schematic of data center backbone showing full mesh logical overlay network

center backbone network, where additional layers of switches (up to 2) further aggregate and distribute traffic. Various designs have been proposed for the backbone network: fat-tree [17], ring [18], etc. Zeppelin makes no assumptions about the backbone network design. The only requirements Zepplin makes on the network are:

- the the first layer of aggregation (TORS-equivalent) is connected by some kind of overlay network in a full mesh, which may or may not be MPLS, see Figure 1,
- the source-side TORS can route an MPLS-labelled packet into the overlay tunnel for the destination-side TORS,
- the destination-side TORS can route an MPLS labelled packet to the virtual switch (VS) and server designated by the label,

Additionally, the source TORS may have multiple overlay tunnels available to a particular destination TORS, which would allow the TORS to do multi-path routing. In the following discussion, we assume that the TORS are OpenFlow-capable MPLS switches and that the overlay mesh between the TORS is also MPLS.

The Zeppelin data center network virtualization scheme is based on an OpenFlow [19] control plane and an MPLS data plane [9]. Although the system was implemented using OpenFlow 1.1, later versions of OpenFlow that support MPLS could also be used. The basic approach involves having an OpenFlow-capable virtual switch (VS) on the server tag packets from a source VM with a label encoding a tenant id and a label specifying the *destination* link between the destination TORS and the destination VS on the server where the destination VM resides. While MPLS labels as used in wide area networks typically have no meaning outside the router and link on which they appear, here we are repurposing them as a way to route to a specific destination link across the data center, rather than using the IP address or MAC address to route to a specific destination node. The OpenFlow controller maintains the mapping between the destination IP/MAC address and the destination MPLS link label.

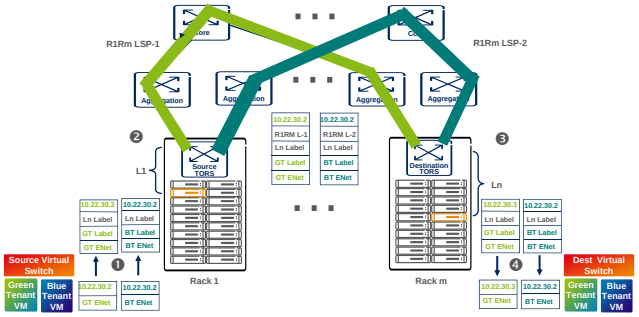

Fig. 2. Basic unicast routing scheme

In the following sections, we describe Zepplin's design for virtualizing the network for unicast routing and how Zepplin handles VM movement.

A. *Unicast Routing Scheme*

Figure 2 illustrates the basic unicast scheme with a simple example. In the figure, there are two tenants, the Green Tenant and the Blue Tenant. Both tenants have VMs running on the same servers, one server is in Rack 1 and the other in Rack m. The IP address of the two VMs running on the same server in Rack m is exactly the same, 10.22.30.2, though the MAC addresses differ. The link between the source TORS and the source server on Rack 1 is assigned label L1, while the link between the destination TORS and the destination server on Rack m is assigned the label Ln. Two LSPs are available between the source TORS and destination TORS: R1RmLSP-1 and R1RmLSP-2. This example was not chosen because it is representative of an actual deployment, but rather to illustrate that Zeppelin handles overlapping IP address spaces correctly, ensuring tenant isolation.

Routing proceeds through the following steps (list numbers correspond to numbers in the figure):

1) The Green Tenant and Blue Tenant VMs emit a packet bound for their destination VMs in Rack m. The source VS pushes tenant labels on the packets, indicated by GT Label for the Green Tenant and BT Label for the Blue Tenant, and a label for the destination link, indicated by Ln Label.
2) The source TORS uses ECMP [20] to choose a different path for each tenant and pushes inter-TORS routing labels onto the packets. Packets destined to the Green Tenant destination are routed over the LSP R1RmL-1 while those destined to the Blue Tenant are routed over the LSP R1RmL-2.
3) The packets are routed through the overlay to the destination TORS. The destination TORS pops the inter-TORS labels and inspects the link label. The link label indicates that the destination is on link Ln, so the destination TORS routes the packet out the corresponding physical port.

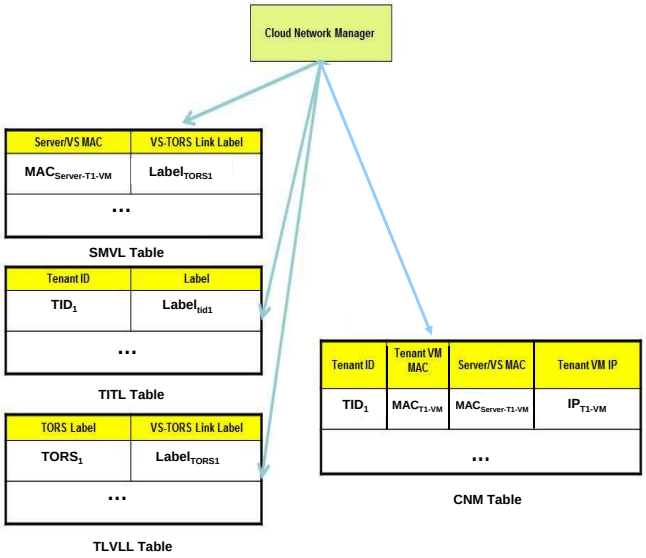

Fig. 3. Tables in the cloud network manager

4) The tenant label and destination IP address are used to determine which VM gets the packet. The destination VS pops the link label and tenant label and delivers the packet to the appropriate VM.

To set up the routing, the OpenFlow tables in the source and destination VS and the source and destination TORS must be programmed. The following sections describes how Zeppelin interposes on common IP network operations to achieve this, and the changes required to the cloud operating system to support Zeppelin.

B. Changes in Cloud Operating System

The cloud operating system typically has a Cloud Execution Manager (CEM) that manages the operations necessary to start, terminate, and move a VM, and a Cloud Network Manager (CNM) that handles connecting a VM to a virtual network and to the Internet. For example, the CEM corresponds to Nova and the CNM corresponds to Neutron in the OpenStack operating system [21]. We assume that the cloud operating system provides service for multiple tenants, and that each tenant is identified by a tenant id.

The CNM maintains mappings between tenant ids, TORS labels, and the MAC and IP addresses of VMs and servers. Figure 3 illustrates the collection of tables implemented in the CNM that maintain these mappings.

These tables and their use are:

- Server MAC to VS Link table (SMVL table) - Look up the link label for the link between the TORS and a VS running on a server in the rack aggregated by the TORS using the server MAC address as a key,
- Tenant Id to Tenant Label (TITL table)- Look up the tenant label using the tenant ID as a key,
- TORS Label to VS Link Label table (TLVLL table) - Look up the link label for the link between the TORS and a VS running on a server in the rack aggregated by the TORS using the TORS backbone mesh label as the key,
- CNM mapping table - Maintains the mapping between the tenant id, tenant VM MAC, tenant VM IP address, and the server MAC address. There is one CNM mapping table per tenant.

These tables are used to set up and manage the configuration of OpenFlow routing in the data center.

C. TORS Flow Table Rules for Packets Incoming to the Rack

OpenFlow 1.1 supports multiple tables and we make use of them in Zeppelin. Zeppelin uses two tables for incoming packets, one to handle the inter-TORS routing labels and one to handle the intra-rack link labels. For packets incoming to the rack, Table 1 is programmed with one rule per overlay connection in the inter-TORS backbone network. This rule matches the inter-TORS back bone overlay routing label. The corresponding action pops the label and sends the packet to Table 2. Because the number of racks in a data center may number in the thousands, to increase scalability these labels can be installed at the time an inter-TORS overlay tunnel is needed. In this way, the limited flow table space is not used up by rule/action pairs unless there is a VM actively using the tunnel.

In the second flow table, a rule/action pair forwards the packet to the appropriate link on the rack. Since the number of servers in the rack is limited (usually 20-40), these rule/action pairs are installed when the rack is initialized. The rule/action for the flow table entry is:

- If the top MPLS label matches the label for a particular link between the TORS and the virtual switch on the rack, forward the packet out the port connected to the link.

These ensure that incoming packets are forwarded onto the correct link.

D. TORS Flow Table Rules for Packets Outgoing from the Rack

For routing outgoing packets, Table 1 in the source TORS is programmed with a rule/action pair that processes the packet through an ECMP Group. Figure 4 shows that part of the TORS Table 1 flow table associated with outgoing packet routing. The flow table itself contains entries with a rule matching the link labels for links between the destination TORS and destination server/VS. If a packet's header matches one of these, then the packet is sent to an ECMP group containing actions that prepare the packet for routing into one of the LSPs leading to the destination TORS. The group hashes the packet header then selects an action bucket based on the hash. The actions in the action buck push a label for one of the inter-TORS back bone overlay LSPs, then forward the packet out one of the physical ports toward the destination TORS. Because there could be up to a hundred thousand destination VS/server links in the data center, these flow rules are only installed when a VM requests a connection with another VM or a destination outside the data center on the Internet.

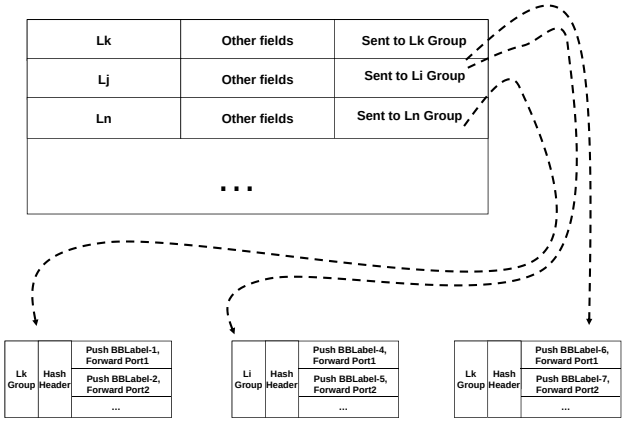

Fig. 4. TORS flow table rules for outgoing packet routing

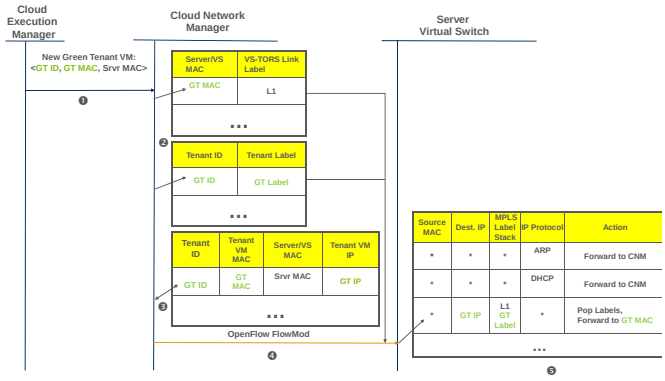

Fig. 5. Message sequence for activating a VM

E. Interventions on IP Network Operations

The virtual switches on the servers need to be programmed with OpenFlow rule/action pairs to intervene on IP address lookup and IP address configuration. There are two rule/action pairs per virtual switch:

- If the protocol type matches ARP, forward the packet to the controller. This ensures that a VM starting a session obtains the IP address information for the destination VM from the controller and allows the controller to install OpenFlow rules for routing between the two VMs.
- If the protocol type matches DHCP, forward the packet to the controller. This ensures that the controller knows the IP address of all VMs in the data center so it can perform the ARP intervention described above. This rule is only installed if the CEM allows tenants to use DHCP; otherwise, the CEM injects the IP address at the time the VM is started, and records the address without requiring an OpenFlow rule.

F. VM Activation

Figure 5 illustrates the message sequence involved in activating a VM on a server. At (1) , the CEM sends a message to the CNM including the Green Tenant ID (GT ID), Green Tenant MAC (GT MAC), and MAC of the server (Srvr MAC) on which the VM will be instantiated. The CNM uses the Srvr MAC as a key to look up the link label of the VS to TORS link in the SMVL table and the tenant ID as a key to look up the tenant label in the TITL table at (2) . At (3), the CNM inserts an entry into the CNM mapping table for the VM, recording the tenant id, VM MAC address, and server MAC address, and the VM IP address if the CEM uses address injection to configure the IP address. If the CEM uses address injection, these are then used together with the Green Tenant IP addresses to program the OpenFlow table in the VS with the following rule/action pairs for handling inbound traffic (5):

- If the first MPLS label matches the VS-TORS link label for this VS, the second MPLS label matches the Green Tenant label, and the destination IP address matches the Green Tenant VM IP address, pop the MPLS labels and forward the packet to the Green Tenant VM's MAC address.

This rule ensures that traffic to the VS incoming on the VS-TORS link is forwarded to the Green Tenant VM if the packet has the VM's IP address.

If the CEM does not use address injection, the inbound traffic rule is installed when a DHCP request is made for an address, described in the next section.

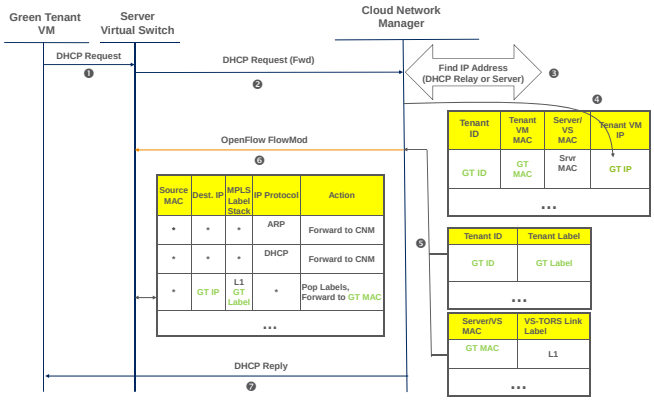

Fig. 6. Message sequence for address configuration

G. VM Address Configuration

If the tenant VM uses DHCP, Figure 6 contains the message sequence between the VM, VS, and CNM. The Green Tenant VM sends a DHCP Request message which is intercepted by the server VS (1). The VS forwards the message to the CNM (2). The CNM then acts as either the DHCP server or as a DHCP relay, finding an address for the VM (3). When the address is found, the CNM records the address in the CNM mapping table (4). The CNM then looks up the Green Tenant label and the link label for the TORS-VS link at (5). At (6), the CNM programs the OpenFlow flow table in the VS for the inbound routing rule/action as described in the previous subsection. The CNM then replies to the VM with a DHCP Reply, containing the address (7).

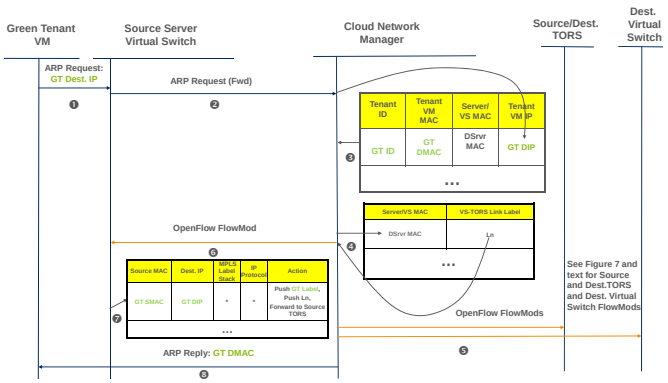

Fig. 7. Destination MAC address discovery message sequence

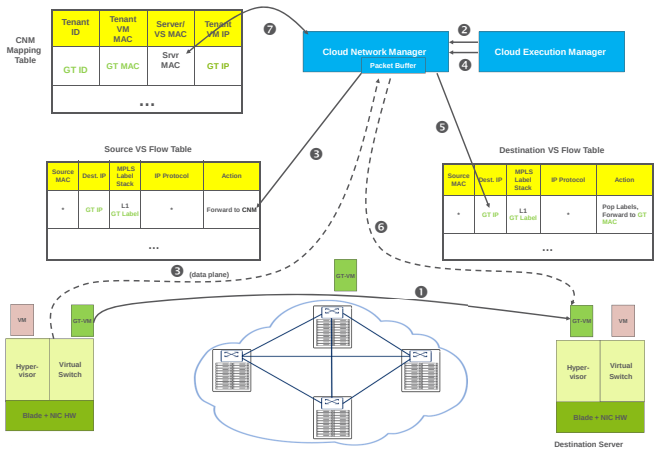

Fig. 8. VM movement

H. Destination IP and MAC Discovery

When a VM needs to contact another VM at an IP address, it will send an ARP message to discover the MAC address associated with the IP address. Figure 7 shows the message sequence triggered by the Green Tenant VM's attempt to discover a destination MAC address. The Green Tenant broadcasts an ARP Request message with the destination IP address (1). The VS intercepts the ARP Request and forwards it to the CNM (2). The CNM looks up the destination IP address in the Green Tenant CNM mapping table, and extracts the Srvr MAC on which the destination VM is running (3). The CNM uses the destination server MAC address to look up the destination VS link label in the SMVL table (4).

The CNM then programs the source TORS with flow routes for incoming and outgoing traffic and the destination VS with flow routes for outbound traffic from the destination VM to the source VM (5), for those rules that were not installed at the time the TORS and VS were started. The TORS flow routes are described Sections III-C and III-D. The flow routes installed in the source VS are shown at (6). The rule/action pair is:

- If the MAC matches the Green Tenant source VM MAC and the IP address matches the Green Tenant destination VM IP address, push a Green Tenant label and Ln - the label for the link between the destination TORS and destination VS/server - onto the packet and forward the packet to the source TORS.

The forward action to the source TORS is shown in in (7). Finally, the CNM replies to the requesting Green Tenant VM with an ARP reply resolving the IP address to the MAC address of the destination VM (8).

I. VM Movement

One of the most difficult aspects of data center networking is dealing with VM movement. When such a movement occurs, the routes to the old server location are no longer valid. If the VM is moved into a new IP subnet, the IP address of the VM can become invalid and existing clients can lose connectivity. Figure 8 illustrates the process of VM movement. The solid arrows indicate control plane messages while the dashed arrows indicate data plane traffic. The numbers are keyed to the list below.

1) The CEM decides on a server and begins moving the Green Tenant VM. As long as the source VM is still capable of taking traffic, no routing changes are made.
2) When the source VM can no longer take traffic, the CEM notifies the CNM.
3) The CNM modifies the source VS flow table to cause any further traffic to the source Green Tenant VM to be forwarded to the CNM for buffering.
4) When the Green Tenant VM on the destination server is ready, the CEM notifies the CNM, or if the hypervisor supports gratuitous ARP, the VM sends an ARP which is forwarded to the CNM through the ARP rule in the destination server VS.
5) The CNM installs a rule into the destination server VS flow table to forward packets with the Green Tenant VM IP address and Green Tenant MPLS label to the Green Tenant VM. If there are no other VMs on the source TORS rack utilizing the link entries for outgoing and incoming traffic that the Green Tenant VM utilized, these are deleted to conserve TORS flow table space, (not shown).
6) The CNM sends any buffered packets to the Green Tenant VM on the destination server.
7) The CNM updates the CNM Mapping table entry for the VM's server MAC address with the new server's address. After this change, any new sessions to the Green Tenant VM's server will be answered by the new VM.

There are still flow table rules on VSs throughout the data center routing traffic on existing sessions to the old server. The CNM uses lazy update to change these rules to the new server. When a packet for the old VM arrives at the old server's VS, the VS encapsulates the packet and forwards it to the CNM since the rule for the old VM has been deleted. The CNM uses the tenant label on the packet to determine the tenant id and the IP address of the correspondent's VM to locate the VS with a rule pointing to the old server, and changes the flow rule on the correspondent's VS and TORS (if necessary) to forward traffic to the new server. The packet is then sent

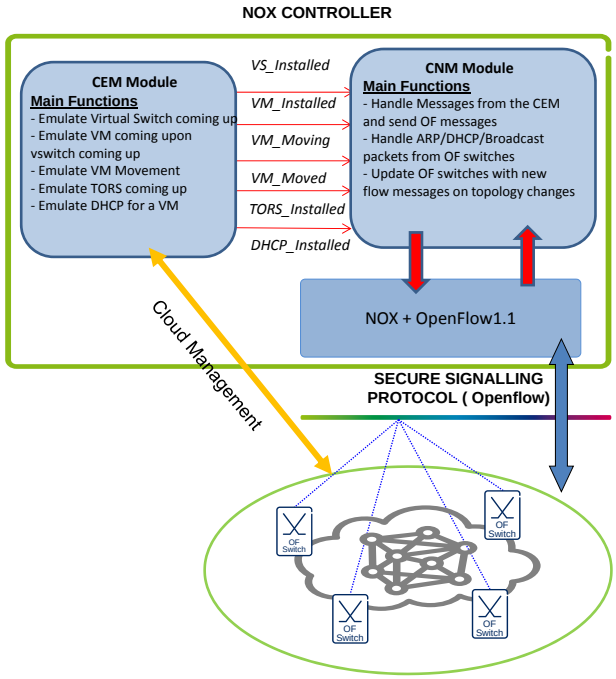

Fig. 9. CEM and CNM implementation

to the Green Tenant VM on the new server. By using lazy update, the CNM is saved from having to update all the rules on every VS at the time of VM movement. All flow rules ultimately either time out or are modified to send traffic to the new server. The CNM can allow the flow rule on the old VS to time out after a grace period, say half a day. VM movement events are not frequent enough that this should prove burden on the VS flow table storage.

IV. Implementation on MiniNet Emulator

We used NOX [22] as the OpenFlow controller and implemented the CNM and CEM modules as applications running over the NOX middleware. As illustrated in Figure 9, the CEM and CNM modules talk to each other through well-defined messages. The message names are descriptive of their function, for example, *VM_Installed* is sent to the CNM when a VM is instantiated. The CNM module, as explained in Section III, is responsible for sending OpenFlow messages to the switches when it receives a notification from the CEM module.

We used Mininet [23] to emulate the data center network. The topology of the network was described through a JSON [24] file. This file was parsed by MiniNet to set up an emulated data center network. We modified MiniNet to store node metadata for every switch/host that was bought up on parsing the topology JSON. We added APIs to MiniNet to help set up a DHCP server. We also emulated VM movement through a JSON entry in our topology file. We added timer entries in the JSON file that helped determine the start of and finish of VM movement.

	TORS Bootup	VM Creation	Session Setup	VM Migration		
				Old	New	Modified
Src VS	0	1 or 3	1	-2	+3	1
Dst VS	0	0	1	0	0	0
Src TORS	k	0	0 or 1+2r	0	0	0 or 1
Dst TORS	0	0	0 or 1+2r	0	0	0 or 1

TABLE I
Summary of rules installation

The CNM module was implemented to have an east-west interface from the CEM module and a north-south OpenFlow interface to program the OpenFlow switches. We described the CEM-CNM interface in the section above. The OpenFlow interface consisted in functions to handle the OpenFlow PACKET-IN message for ARP, DHCP, and broadcast packets, together with a flow removal and join handler. Functions to program the TORS and VS with the rules described above were also implemented.

V. Analytical and Emulated Results

In this section, we present our analysis results to show the scalability of Zeppelin. We are particularly interested in flow table state scalability, since that is a key issue for OpenFlow. We first summarize the rule updates in different scenarios and then discuss the dynamics using different experiments.

Table I shows the number of rules that need to be installed in four different scenarios. When starting a TORS, one needs to install one rule for each server connecting to this TORS. Therefore, assuming that each TORS can support k number of servers, then k rules are installed. Each rule matches a VS-TORS label with the action of forwarding the packets to the corresponding server. No rules are needed in other switches. When a server is started, one and possibly two rules are installed on the VS to forward the ARP and DHCP packets to the controller (only the ARP rule is needed if the CEM configures IP addresses with address injection). These two rules are shared across all the VMs on the same VS. Therefore, they only need to be programmed once for all the VMs on a server. In addition, a rule is needed matching the VS-TORS label, the tenant label, and the VM's IP address, with the action of either popping the label or forwarding the packet. Although this rule contains three logically different actions, it can be implemented as one rule physically. Thus, we have 1 rule on the source VS. This rule is established either when the VM is started, if address injection is used, or when the IP address is obtained through DHCP.

We propose to set up rules dynamically upon VM communication, which will significantly reduce the memory consumption on both the TORS and the VS. Upon a new session established between two VMs, a set of rules will be programmed on the fly. First, one needs to install a rule on the VS to match the destination MAC with the action of pushing the tenant label and the destination VS-TORS link label, as well as forwarding to the source TORS. Second, the source TORS will need a rule to match the destination VS-TORS label and forward to the correct path. But if there is already a session between

any VM connected to this source TORS and the destination VS, this rule can be eliminated. In the forward direction, a separate rule is used to map the path from the source TORS to the corresponding forwarding next hop. Assuming multi-path routing is supported, then the TORS needs to have r number of rules for mapping r ECMP groups. Similarly, on the reverse direction, r rules are installed to match the inter-TORS label and pop the label. Note that these rules may not be needed if there are already sessions established between the two TORS.

Finally, when a VM is migrated to a different server, the controller will modify the inbound forwarding rule to forward the packet to the controller, instead of going to the VM. The corresponding new rule will be installed on the new VS. The rules on the TORS are updated on demand. Triggering by any packets sent to the old VM location, the controller will update the TORS with the new TORS location.

Next, we use experiments to show the rules needed dynamically. We assume a tree topology with 12 racks, which is also used in [25]. Each TORS is connected to 20 servers. Next, we simulate the process of VM creation and inter-VM communication. In the experiment, we continuously start VM instances. Each VM selects n number of other VMs to set up sessions, where n is a random number uniformly drawn from a pre-defined average value. In Figure 10 and Figure 11, we show two cases, *i.e.,* average 5 sessions per VM and average 10 sessions. For each case, we compute the average number of rules per VS and per TORS, as a function of the number of VMs per VS/TORS. From Figure 10 we observe that the rules are linearly increasing as the number of VMs on the VS. However, for rules in TORS, the increasing trend slows down when the number of VMs are large, since many rules only need to be installed once between a pair of VSes and TORSes.

Flow scalability is an important criterion when considering the applicability of OpenFlow. Curtis, et. al. [26] discuss the flow scalability of OpenFlow. They show that a typical TORS of the current generation can support 1500 OpenFlow rules in TCAM, whereas an average TORS has roughly 78,000 flows. In our scheme, flows heading to the same server share a forwarding rule on the TORS, so strategic placement of VMs can substantially reduce the amount of flow table space. In addition, as shown in Figure 11, the number of rules at 5 flows per VM is around 1500, while the number at 10 flows per VM is around 2500 for 1000 VMs per rack. The former number would be well within the current generation TORS capacity to handle, while the latter should be within the next generation. Finally, the analysis done by Curtis at. al. assumes the flow rules are handled in the TCAM. Many switch chips today contain support for MPLS, so a carefully optimized implementation of OpenFlow on such a chip could potentially avoid having to use the TCAM and might support even more rules.

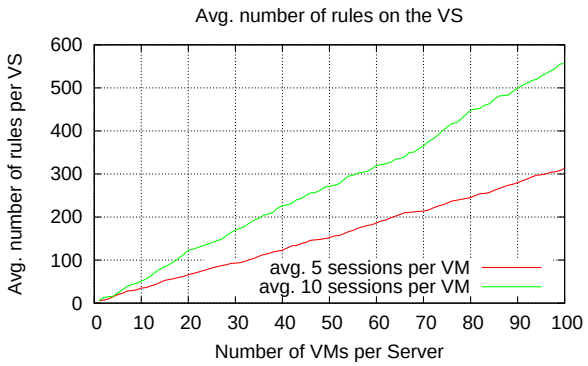

Fig. 10. Average number of rules per VS

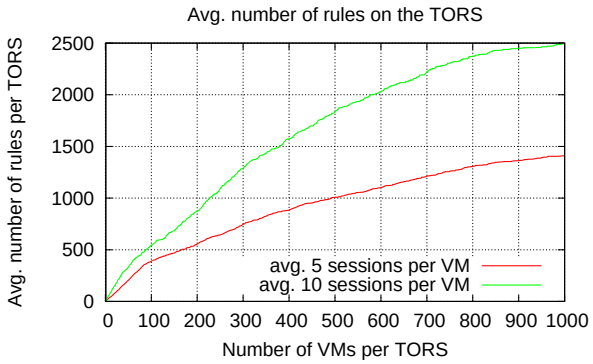

Fig. 11. Average number of rules per TORS

VI. Conclusion

In this paper, we have described the design and implementation of Zeppelin, a third generation data center virtualization scheme based on MPLS labels. Zeppelin uses two layers of MPLS labels: one identifying the virtual network upon which the tenant is located and one identifying the routing path through the network. The routing path layer itself consists of two labels: one identifying the link between the destination last hop virtual switch and last hop top of rack switch, and one identifying the route through the back bone network. In contrast to the use of MPLS in wide area networks - where MPLS labels are meaningless outside of the link and router on which they are allocated - in Zeppelin, the MPLS labels have semantic relevance across the data center. A centralized controller handles assignment of MPLS labels and programming of virtual switches and top of rack switches using the OpenFlow protocol.

An implementation of the scheme was built using NOX and tested on top of the MiniNet emulator. An analysis of the emulation shows that the flow rule scalability is within the limits of existing switch chips if TCAM is used to hold the rules, and could even be more scalable if the OpenFlow implementation utilizes the built-in MPLS support of the switch chips.

For future work, we have planned an extension of Zeppelin to multicast, and an implementation on actual OpenFlow hardware. We also plan to study actual data center traffic to see if Zeppelin can efficiently support it.

VII. Acknowledgements

The authors would like to thank Ponnanna Uthappa from USC for coding the NOX implementation.

References

[1] "Amazon Elastic Compute Cloud." http://www.aws.amazon.com/ec2.
[2] "Google App Engine." http://www.code.google.com/appengine/.
[3] "Windows Azure Platform." http://www.microsoft.com/windowsazure/.
[4] "The Rackspace Cloud." http://www.rackspacecloud.com.
[5] "Cloud Computing." http://www.GoGrid.com.
[6] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation (GRE)." RFC2784, 2000.
[7] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." Internet Draft, 2012.
[8] "Networking in the Era of Virtualization." http://nicira.com/sites/default/files/docs/.
[9] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta, "MPLS Label Stack Encoding." Internet Engineering Task Force, RFC 3032, January 2001.
[10] "IEEE Std. 802.1Q-2011, Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks." IEEE, 2011.
[11] "SDN." http://en.wikipedia.org/wiki/Software-defined_networking.
[12] D. Allen and N. Bragg, "802.1aq shortest path bridging: Design and evolution." IEEE Press, 2012.
[13] "Contrail: The Juniper SDN controller for virtual overlay network." http://www.juniper.net/us/en/dm/junos-v-contrail/, 2012.
[14] "OpenFlow." http://www.openflowswitch.org.
[15] D. Allan, J. Farkas, P. Saltsidis, and J. Tantsura, "Ethernet routing for large scale distributed data center fabrics," in *Proceedings of the IEEE CloudNet*, 2013.
[16] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, "Carving research slices out of your production networks with openflow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, 2010.
[17] A. L. Mohammad Al-Faris and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM Proceedings*, 2008.
[18] K. T. L. S. Y. Z. Chuanxiong Guo, Haitao Wu and S. Luz, "Dcell: A scalable and fault-tolerant network structure for data centers," in *SIGCOMM Proceedings*, 2008.
[19] "Openflow switch specification: Version 1.1.0 implemented (wire protocol 0x02)," Feburary 2011.
[20] C. Hopps., "Analysis of an equal-cost multi-path algorithm." Internet Engineering Task Force, RFC 2992, November 2000.
[21] "Openstack." http://www.openstack.org.
[22] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, July 2008.
[23] "Mininet." http://mininet.org/.
[24] "JSON." http://www.json.org/.
[25] R. N. Mysore, A. Pamporis, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, Subramanya, and A. Vahdat, "PortLand: A Scalable, Fault-Tolerant Layer 2 Data Center Network Fabric," in *SIGCOMM 2009*.
[26] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 254–265, August 2011.