Cloud Rack: Enhanced Virtual Topology Migration Approach with Open vSwitch

Yan Pu, Yilong Deng, Aki Nakao
Graduate School of Interdisciplinary Information Studies,

The University of Tokyo
Tokyo, Japan
yan@nakao-lab.org yilongdeng@gmail.com

Abstract—**Cloud computing has emerged to provide infrastructure for cost-effectively hosting applications with dynamically changing demands for computational resources. Dynamic resource allocation in cloud computing is made possible through virtualization technologies. In addition, live migration of services hosted in virtual machines (VMs) among multiple data centers is expected to add more flexibility in elastic resource allocation. Our recent work, Mobitopolo, has shown the proof of concept of an infrastructure to enable live migration of VMs with logical connections intact among them. However, since Mobitopolo makes more of portability, it suffers from network performance. In this paper, we propose alternative design and implementation of logical topology migration using KVM and Open vSwitch (OVS) to realize an infrastructure for applications demanding for much network performance.**

Keywords-Cloud Computing, network virtualization, live migration, Openflow

I. INTRODUCTION

As one of the key technologies in so-called cloud computing, virtualization has recently become a popular research topic. Virtualization of both operation systems and networks is under discussion. Virtualization of operating systems mainly provides elastic allocation of processing and storage resources, dynamic deployment and migration of applications, and optimization of power consumption. Virtualization of networks offers dynamic construction and rearrangement of private network topologies [4]. Combining these two technologies provides possibility of enabling a new class of distributed systems on top of VMs that may live-migrate anywhere while maintaining the logical connection among them. Our recent work of Mobitopolo [2] has provided the proof of concept of the live migration of a set of VMs while the logical connection among them is preserved.

The migration of a private network of VMs requires both live-migration of VMs and that of their virtual network topology right after that. In Mobitopolo, the whole process is implemented on User-Mode-Linux (UML) and Ethernet-over-UDP. Portability is the main focus, so that the migration of a network of VMs may be enabled anywhere where the underlying host operating systems are a variant of Linux. However, since it is implemented in user-space with transport layer tunnels, its performance suffers considerably especially in terms of network I/O.

In this paper we propose Cloud Rack that takes another implementation approach to the concept of Mobitopolo with much higher performance. Cloud Rack is built on Kernel-based Virtual Machine (KVM) [7] connected through L2 tunnels enabled by Open vSwitch (OVS) and GRE-over-Ethernet protocols. We implemented Topology Central Control Service (TCCS), a central control unit that works with NOX [6] to provide the ability to manage all the virtual links so that their reconfiguration can be done automatically after migration. The whole process is also transparent to all the VMs. Besides all the benefits that Mobitopolo provides, our performance evaluation shows that Cloud Rack gains much higher performance than Mobitopolo.

The rest of the paper is organized as follows. Section 2 describes the architecture design. Section 3 briefly introduces how TCCS works with NOX for tunnel migration. Section 4 then shows micro-benchmark results. Finally, Section5 discusses our immediate future work and concludes.

II. ARCHITECTURE DESIGN OF CLOUD RACK

As mentioned above, Cloud Rack has three main parts in its architecture, VMs, OVS and TCCS. The whole architecture is illustrated in Figure 1. We adopt KVM for a virtualization mechanism in our architecture. A virtual network device in a KVM guest is connected to a TAP device in the KVM host, and the TAP device is then connected to an OVS. The OVS provides GRE-over-Ethernet tunnels. In our architecture design, tunnel migration is performed by locating each VM before and after the migration. Tunnels can be properly setup among VMs and flows can be rearranged by OVS after the tunnel setup.

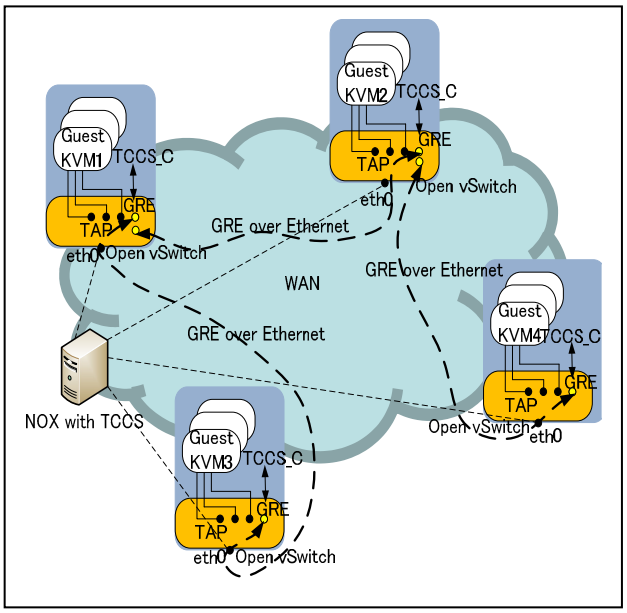

Figure 1: Cloud Rack's architecture

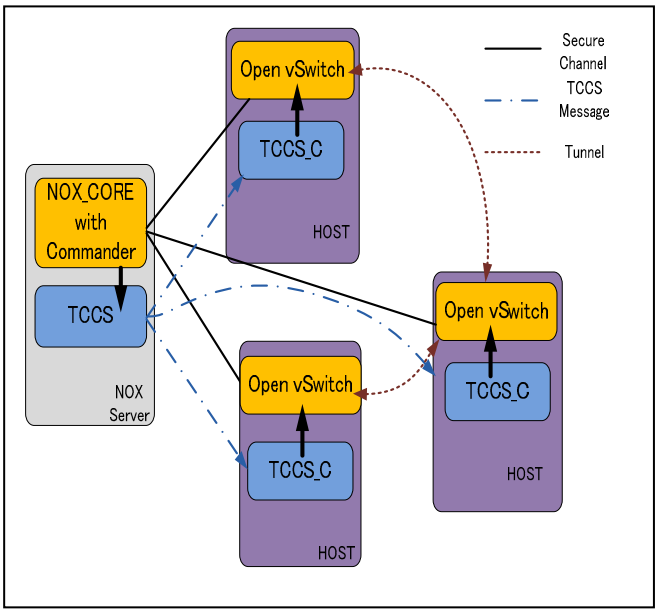

Figure 2: Abstract Structure of TCCS

In our implementation, TCCS locates all the VMs through Python Twisted and arranges flows by an applet of NOX. It sets up tunnels using bash scripts of OVS commands.

Our architecture design has several advantages in achieving higher performance than Mobitopolo. First, both VMs and tunnels implemented in kernel-mode. Second, since OVS uses L2, reducing overhead compared to UDP-tunnel processing in Mobitopolo. Third, all the flows may be arranged on demand instead of preparing the full mesh topology in advance. For example, when KVM1 sends packets to KVM4 in Figure 1, a new tunnel may be setup immediately between them. Tunnels can be released when it is idle.

III. Topology Central Control Service for Tunnel Migration

In our architecture, Topology Central Control Service (TCCS) is in charge of locating each VM so that logical connections (tunnels) among them can be automatically reconfigured after VM's migration. In this section, we briefly introduce TCCS and then present the migration of two VMs with a single tunnel between them as a simple example to show how TCCS works.

TCCS adopts the client and server architecture where the server side runs a modified NOX server that can take not only OpenFlow commands but also our proprietary ones to handle migration, while the client side runs a modified OVS likewise to deal with migration.

An OVS registers flows to a NOX server and learns what action should be applied for the flows. Thus, we create an applet named Commander for NOX to command all the OVS's to perform L2 forwarding. Especially, to deal with packets to be forwarded from one host to another over WAN, a tunnel between them should be established.

To set up a tunnel, the Commander firstly calls up TCCS on the NOX server, and then TCCS sends commands to the corresponding TCCS clients to set up tunnels. A command from TCCS commonly includes the MAC address and IP address of a host that will be tunneled to. TCCS clients parse the information in commands and run a script with the information to create the tunnel. The communication between TCCS and the clients is depicted in Figure 2.

Each TCCS client also registers its host's IP address and its OVS's *datapath identifier (dpid)* to TCCS so that TCCS may keep the information of each host's IP address associated with its OVS's *dpid*. The *dpid* of each OVS is a 16-bit hex number where the lower 12 bits are equal to those of the host's MAC address. When a tunnel is needed, TCCS gets a message with the dpids of two OVS's involved from Commander. Then TCCS obtains the hosts' IP addresses and MAC addresses and send them to the two hosts' TCCS clients. The TCCS client will then execute a script to create a tunnel port and insert it into its OVS.

Now we show how TCCS performs tunnel migration. In our example, we have two VMs and a single tunnel between them. The VMs are originally located in two hosts named Source1 and Source2. The task is to let the two VMs migrate to the other two hosts named Dest1 and Dest2 while preserving the tunnels between the two VMs. The task is illustrated in Figure 3.

In our example, TCCS clients on the four hosts are registered to TCCS in advance. After KVM1 and KVM2 migrated to the Dest1 and Dest2, and when the processes on the two KVMs try to continue their communication between them, two OVS's on the destination hosts first forward the unmatched requests to the NOX server, Commander then decides that the flow should be forward via a tunnel port on each OVS. Commander will also check all the port information on each OVS.

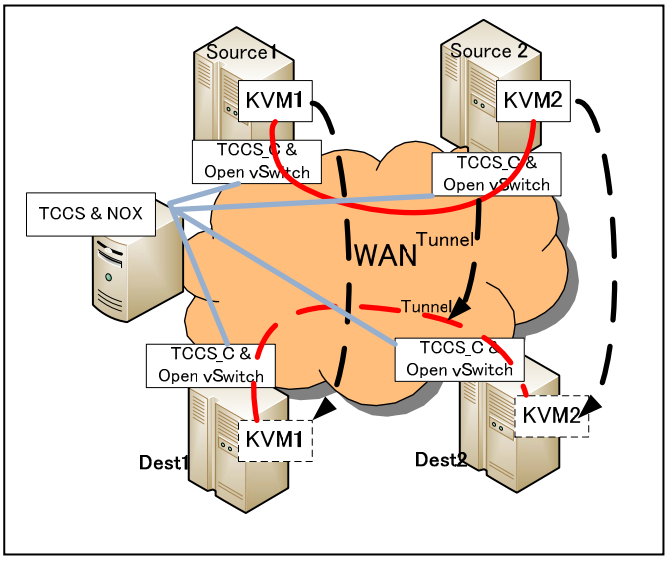

Figure 3: Example Task of TCCS

If it discovers that the necessary tunnel port doesn't exist on each OVS, Commander then gives the two dpids of OVS to TCCS. TCCS will then arrange two TCCS clients on the two destination hosts to bring up the tunnels as described earlier.

The simple walk-through example given above illustrates how TCCS performs tunnel migration. However, our prototype implementation of TCCS shows that it can properly reconfigure all the tunnels among VMs after migration in more complex setting than the example shown here. Especially, TCCS can live-migrate VMs with heavy network traffic flowing between them.

IV. Experimental Results of Cloud Rack's Performance Test

To explore the performance of Cloud Rack, we carry out live-migration experiments and examine the throughputs in a virtual network layer both statically and during migration. In the following subsections, we describe our experimental environment first, and then discuss the results.

A. Experimental Enviornment

The aims of our experiments are first to examine whether TCCS can properly reconfigure the tunnel after live-migration , then to check how long the whole process takes for the tunnel recovery between VMs. We deploy three servers in two different campuses of the University of Tokyo that are about 5 km away and connected over the Internet. To be more specific, two of the servers are located within the same subnet in one campus and the other server is located within another subnet in the other campus. Between the two campus, network connections are provided by common ISPs. The basic information of the three servers is shown in Table 1.

The experimental environment is shown in Figure 4. Originally, each of Server1 and Server2 hosts a single VM. As described in Section 2, their virtual NICs are connected to TAP devices and the TAP devices are inserted into OVS's. A GRE -over-Ethernet tunnel is established between the two VMs for the communication between them.

TABLE I. INFORMATION OF SERVERS

	Server 1 & 2	Server 3
CPU	Inter Core2 Quad CPU Q6700 2.66Ghz	Intel(R) Xeon(R) CPU X5550 2.67GHz
Memory	4GB	24GB
NIC Type	Intel 82566DM-2 Gigabit	Broadcom NetXtreme II BCM5709 Gigabit
Hard Disk	SATA 7200rpm 8Mb L2 cache	SCSI 7200rpm 8Mb L2 cache
OS	Debian Lenny with Kernel 2.6.26-2-686	Ubuntu 9.10 with Kernel 2.6.31-14-generic

We examine the throughput between the two VMs statically during the first 120 seconds when they are hosted in Server1 and Server2. After that, we check the throughput during another 120 seconds, but letting VM2 migrate to Server3 during this period. The live migration is implemented by KVM and VM2 uses a virtual disk image in a shared NFS directory. We use *iperf* tool in the VMs in our experiments.

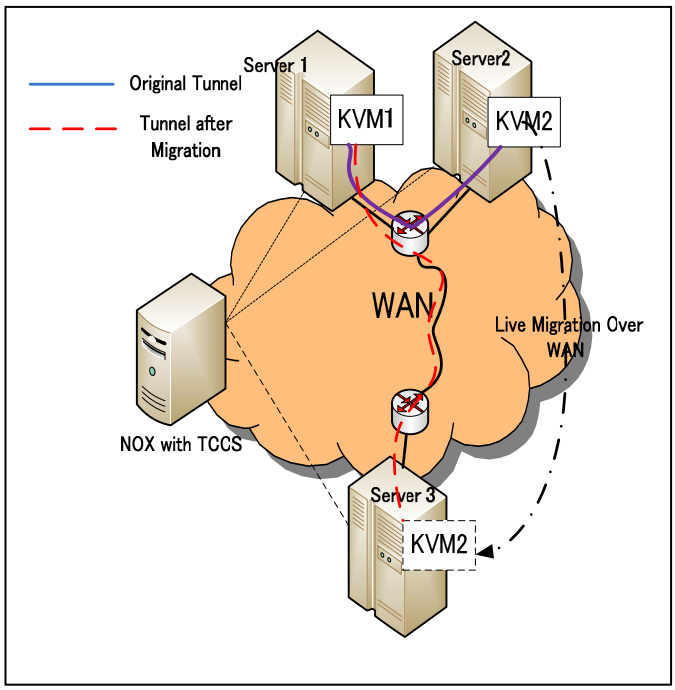

Figure 4: Experiments Environment

An *iperf* client runs in VM1 and sends packets with transmission rate of 100Mbps to an *iperf* server running in VM2. We also record the static throughputs between the two VMs when they are hosted in server2 and server3. Then we plot the results as follows.

B. Experiment Results

Figure 5 and Figure 6 show the static throughput between the two VMs, when VM2 is hosted in Server3 and Server2, respectively. Figure 7 shows the throughput during migration.

We see from these plots that in general the throughput stays the same during the live-migration. When the VM2 migrate from server2 to server3, only 1-2 seconds of interrupt is observed for the tunnel to be reconfigured. We believe that the time of interrupt is tolerable for most network applications.

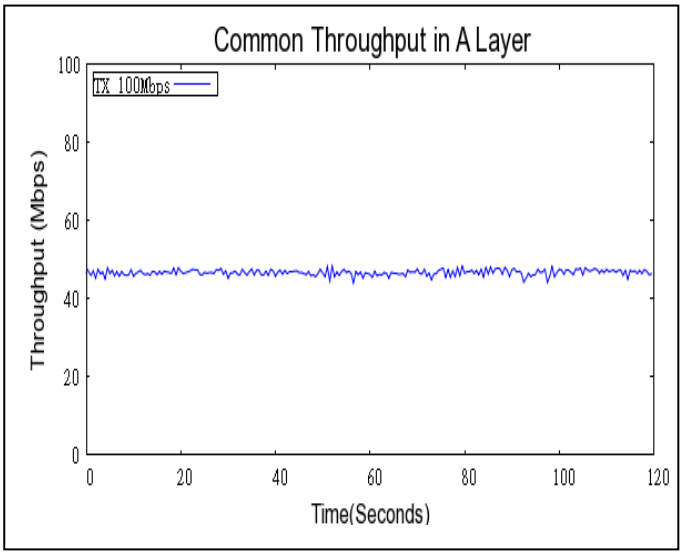

Figure 5: Common Throughput in a Layer
(Hosted in Server 1 & Server 3)

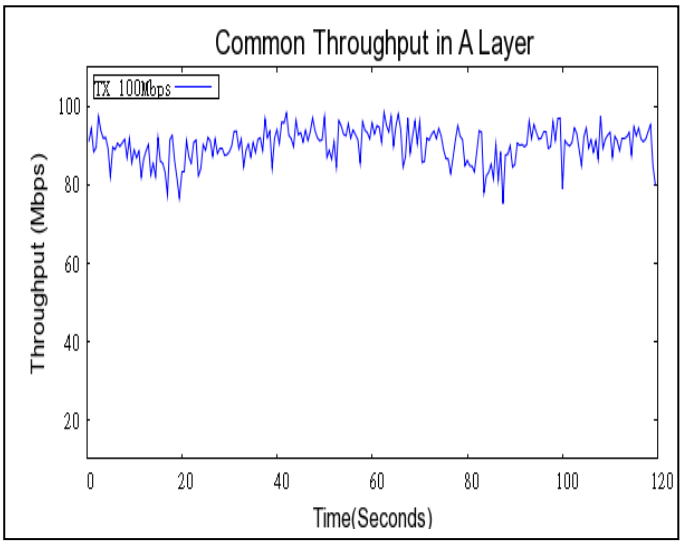

Figure 6: Common Throughput in a Layer
(Hosted in Server 1 & Server 2)

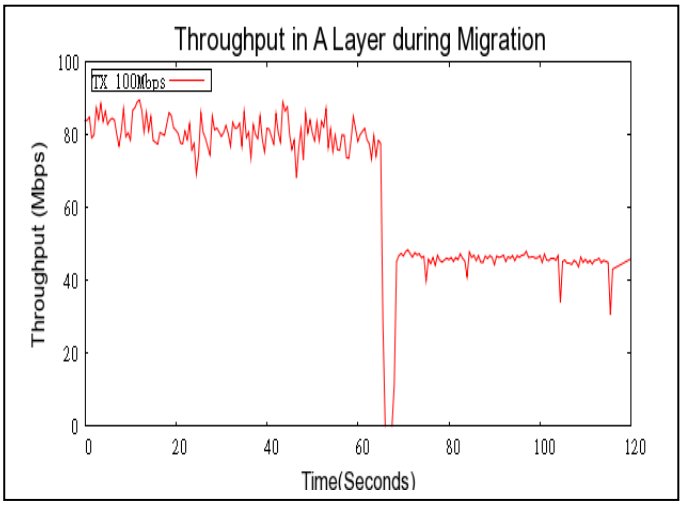

Figure7: Throughput in a Layer during Migration
(VM2 Migrated from Server 2 to Server 3)

V. Future Work and Conclusion

We believe that Cloud Rack is a viable solution for providing portable architecture over WAN. By adopting KVM and OVS with its GRE-over-Ethernet as the components of our architecture, we gain much higher performance than the existing work. Since the OVS is used as a virtual OpenFlow switch in our architecture, additional functions can be realized by programming on the NOX server. Besides, deployment of Cloud Rack is quite simple that the TCCS itself only needs a Python environment to run.

We implement the prototype of Cloud Rack in such a way that there may be only one virtual network. However, multiple virtual networks may be supported easily. We develop configuration tools of TCCS that supports management of only one virtual layer network. The extension towards multiple virtual networks is our immediate future plan.

In addition, we also plan on more experiments including performance tests of the real network applications under our architecture.

References

[1] Hao, F., et al. , Enhancing dynamic cloud-based services using network virtualization. ACM, 2009: pp. 37 - 44.

[2] Potter, R. and A. Nakao, Mobitopolo: a portable infrastructure to facilitate flexible deployment and migration of distributed applications with virtual topologies. ACM, 2009: pp. 19 - 28.

[3] McKeown, N., et al. , OpenFlow: enabling innovation in campus networks. ACM, 2008: pp. 69 - 74.

[4] Ben Pfaff, et al., Extending Networking into the Virtualization Layer. HOTNETS, 2009.

[5] Sundararaj, A.I. and P.A. Dinda. Towards virtual networks for virtual machine grid computing. 2004: USENIX Association.

[6] Natasha Gude, et al. NOX: Towards an Operating System for Networks. CCR, July 2008

[7] Kivity, A., et al. KVM: the Linux virtual machine monitor. 2007.

[8] Valancius, V., et al. Transit Portal: Bringing Connectivity to the Cloud. 2009: SIGCOMM.
[9] Bavier, A., et al. In VINI veritas: realistic and controlled network experimentation. 2006: ACM.
[10] Pfaff, B., et al. Extending Networking into the Virtualization Layer. 2009.
[11] Wang, Y., et al., Virtual routers on the move: live router migration as a network-management primitive. ACM SIGCOMM Computer Communication Review, 2008. 38(4): pp. 231-242.