2013 IEEE Sixth International Conference on Cloud Computing

# Towards Network Virtualization Management for Federated Cloud Systems

Mon-Yen Luo, Shang-Wei Lin, and Jun-Yi Chen
Department of Computer Science & Information Engineering,
National Kaohsiung University of Applied Sciences, Taiwan
* myluo@kuas.edu.tw

*Abstract*- **The federated cloud system provides the ability to integrate, manage, and use third-party resources from multiple institutions and combine the resources into one infrastructure. A virtual network management system is needed to supporting a seemingly infrastructure where services can be deployed on-demand and across different network domains. We present our work on designing and implementing an integrated system to enable virtual network services across federated clouds. We propose a novel mechanism to enable network stitching via translating VLAN tag dynamically. The service can be partitioned on hosts across different network domains using different underlying layer-2 or layer-3 network interconnection technologies. This paper also provides the results of performance measurement from the implementation based on real production networks.**

***Keywords: Cloud Federation; Network Virtualization; OpenFlow.***

## I. INTRODUCTION

Cloud Computing [1,2] has emerged as an important architecture and concept for the Internet, leveraging virtualization to provide computing, networking resources, and storage on-demand to multiple tenants. It is increasingly agreed that a significant number of services can be delivered using a cloud-computing model in which users access the applications and services hosted in data centers located in various parts of the network from any device with network capability. Such highly distributed data environments have network requirements that are distinctly different from those of general-purpose networks. In particular, in order for a service application to be deployed as part of a cloud, the physical resources should be abstracted away and fully isolated for different tenants. As a result, cloud services are increasingly based on large-amount deployment and rapid configuration of virtual machines (VMs), which has introduced many challenging problems to the field of networking. Cloud networks must support automated managements (creation, deletion, and migration) of enormous numbers of virtual machines, large numbers of attached physical and virtual devices, and isolated independent subnetworks for multi-tenancy (application components belonging to different tenants are collocated on the same physical substrate) [3].

To enhance collaboration, the Ministry of Education of Taiwan funded a project to establish the Teaching/Learning Resources Center of Southern Taiwan (STTLRC for short) in 2008. The main mission of this center is to integrate and share the teaching expertise, knowledge and resources to 37 universities around southern Taiwan. To this end, we have built an education cloud [4] to support academic departments or institutions in creating their own E-learning services, without having to invest in IT infrastructure and staff. Since the learning service is carried out on the cloud, our partner universities do not need to own nor operate the system. The users send their input data to the cloud where it is processed, and the results are then sent back to the users. However, some of our partner universities have strong concerns about data privacy, which has been one of the major obstacles for the acceptance of cloud paradigms. In response, we have enhanced the single-domain cloud system to enable a *federated cloud* model [5]. The federated cloud provides the ability to integrate, manage, and use third-party resources from multiple cloud service providers and combines the resources into one infrastructure. With this federated cloud model, the institutions can join their own resources to the cloud and gain more control over the system since part of the infrastructure is under their control.

The network infrastructure in such a federated cloud system faces many further network challenges. In essence, VMs belonging to the same service may be hosted either on the same host or on a collection of physical hosts across different network domains. Dynamic interconnections among many resources at multiple remote sites on-demand are required in order for these VMs to communicate with each other over a private network. A management system capable of managing both the physical and virtualized network mechanisms is needed to form a coherent infrastructure. A promising concept to address the above issues is network virtualization, whereby several network instances with dynamic topology can be created dynamically on a common physical network infrastructure. This paper presents our work on implementing a middleware system to dynamically build and manage virtual networks on a federated cloud system over production research networks. The specific contributions presented in this paper are the following. First, we present an approach to synergize several important technologies, including virtual switches, OpenFlow, and NetFPGA, into a coherent platform that enables the dynamic configuration and management of virtual networks. Second, we propose a novel mechanism to enable network stitching via translating VLAN tag dynamically. Third, we provide the results of performance measurement from the implementation based on real production networks.

The remaining of this paper is structured as followed: We start out by discussing the background of our work and our federated cloud model in Section 2. We also describe the system requirements as well as our design in section 2. Section 3 and Section 4 present the design and current implementation of the proposed system. We present the performance result in section 5.

We discuss the related work in section 6 and conclude in section 7.

## II. Federated Cloud model and its requirements

This section describes our federated cloud model and derives requirements of the virtual network from its service principles.

### A. *Federated Cloud Model*

The major design goal of a cloud system is to enable that a service can be dynamically created and dedicated to the different tenants upon request. For example, if a university requires a course management system with an ePortfolio system, we can quickly launch an instance provided to them as a service from our federated cloud system. The federated cloud system may encompass a group of loosely coupled resources (such as machines, network devices, storages, and so on) contributed by different infrastructure providers. A tenant service in the federated cloud should be deployed in the form of partitions on multiple infrastructure providers across the Internet. We will use the example in Figure 1 to illustrate this idea. The service described is a typical multi-tiered Internet service, composed of web portal, authentication server, user database, application server and backend database. Traditionally, such a service would be provided by servers in the service provider's data center. Our federated cloud enables the service to be deployed by a number of virtual machines with specific components at different locations.

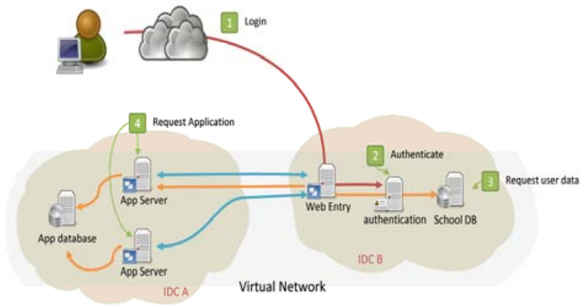

Figure 1: Reference Model of a Federated Cloud

Following the example depicted in Figure 1, the service is provided by component servers partitioned in two locations: the IDC A is the datacenter of our campus; and IDC B is the datacenter of one of our partner universities (NCKU). Such a service partition is transparent to users. The users can access the service in the following usual scenarios: (1) A user logs into the system first; then (2) the system checks his or her identity using some authentication mechanism, then (3) the system gets the user profile from the user database, then (4) the system accesses another learning objects database based on the related user information, and mashes up all the content from components and return it to the users. Supporting our vision of providing services in such a federated cloud model has complications and challenges. We have designed and implemented an integrated system framework to address these issues. In general, the system must support (1) handling requests from service providers regarding resource and topology requirements, (2) finding and allocating available physical system resources, (3) instantiating various virtual machines running specific application components, (4) creating and instantiating virtual networks to interconnect VMs based on the desired topology, and (5) managing simultaneous operations of multiple virtual networks on top of a shared physical infrastructure. The Virtual Network Management layer (step 4 and step 5) is the core contribution of this paper, which will be discussed in detail in the following sections. The interested reader is referred to [5] for details of the rest (step 1 to step 3) of the framework.

### B. *Network Requirements of the Federated Cloud*

As we discussed above, we require a system to create and manage simultaneous virtual networks on demand with arbitrary topologies on a loosely coupled distributed cloud system. How to implement the virtual network across several production networks will be the major challenges discussed in this paper. In our work to address these challenges, we strive for practical approaches that take into account the realities of practical environments. As depicted in Figure 2, we have a complex and heterogeneous network environment to interconnect each data center of our partners. Now, our partner universities are majorly interconnected with shared, public layer-3 networks, i.e. TANet (Taiwan Academic Network) and TWAREN (Taiwan Advanced Research and Education Network). With supports of iCAIR at Northwestern University and NCHC (National Center for High-Performance Computing), our testbeds have also been extended to connect with iGENI projects [6] in the United States.

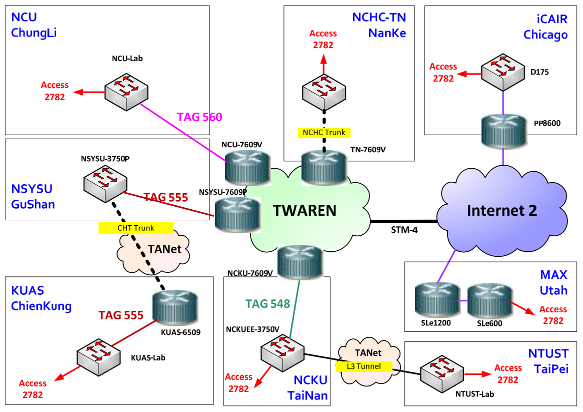

Figure 2: Network Topology of Current Cloud Federation

We require an isolated network mechanism to protect production and the private cloud network from each other. This mechanism is a typical requirement in the federation of cloud systems. In response, some of our partner universities (e.g., NCKU, KUAS, and NCU) applied layer-2 mechanisms (e.g., VPLS [7]) to interlink the private cloud systems. However, some of our partner universities (e.g., NTUST) are unable to have such network facilities to interconnect their network and isolate certain traffic. In these cases, the interconnections are created using a layer-3 over layer-2 mechanism (e.g., IP tunneling mechanism) among these sites. These mechanisms cause a heterogeneous networking environment. Further challenges exist that we must address in this paper. First, each service should have its own private network with customizable topology and independent address space. Our system must provide the capability to dynamically create each network. Second, networks inside each service should be in a closed environment for security, performance, and privacy reasons. VPLS is only a partial solution for these problems. The first reason for this shortcoming is that the use of 12-bit VLAN IDs limits the solution's scalability. In our environment, we are only allowed to use the VLAN 2782 to interconnect all sites, which causes another serious scalability/performance problem, that is, the system suffers from the performance problem caused by flooding mechanisms (e.g., ARP broadcast), huge MAC table size, and the use of Spanning Tree Protocol [8,9]. Third, the VLAN or VPLS does not have an easy management to configure the path across several network domains. Furthermore, these L2 or L3 mechanisms are mainly point-to-point dedicated connectivity, and require a series of interactions and manual operations between network operators and service providers.

### C. System Design

This paper provides a general solution for the above problems, whether our partner sites have layer-2 network isolation capabilities technology over layer-3 networks available or not. This new approach provides a more flexible, controllable programmable path creation and management mechanism. Our approach represents a new layer in the software stack, providing the hooks to other middleware and serving as a control plane to orchestrate all operations. Basically, our system can be divided in to two parts: the intra-cloud and the inter-cloud, as depicted in Figure 3. Each tenant service has its own virtual network with its own particular address space. Each VM in a particular virtual network has a virtual IP address assigned and managed by the Virtual Network Management system. Each virtual network can also be configured with a VLAN tag so that an overlapping address space is allowable.

In one cloud system, we utilize the virtual switches in hypervisor as the building blocks for creating the virtual network. We also use OpenFlow protocol [10] to control and manage the virtual networks. For inter-cloud, we propose a gateway system with a novel VLAN translation mechanism to enable the virtual private network spread over public Internet and more effective usage of VPLS facilities like VLAN tag. The related mechanisms of VLAN translation are implemented in an OpenFlow switch based on NetFPGA. We discuss the design and implementation of the intra-cloud and inter-cloud mechanisms in more detail in the following two sections.

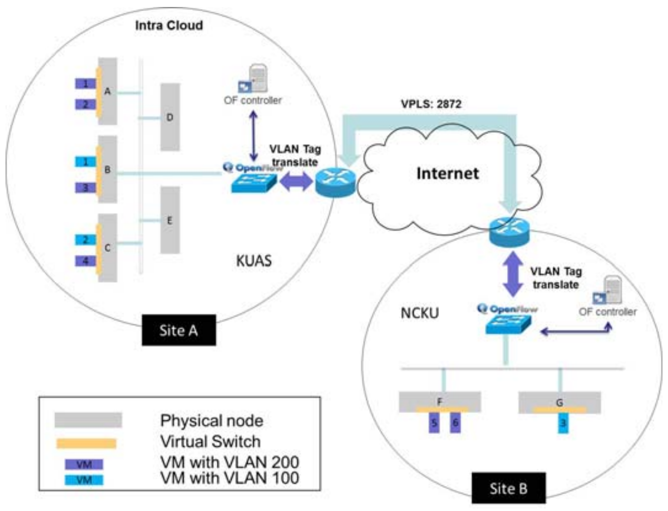

Figure 3: System Overview

## III. Intra-cloud Mechanisms

In this section, we first introduce the virtual switching mechanisms in the hypervisor and its usage in our design. We then describe our implementation of the extension on the virtual switching techniques.

### A. Virtual switching

The major design goal of our system is to enable dynamic connectivity on-demand among multiple virtual machines. As multiple VMs are instantiated on one physical machine, a new networking layer is required on the hypervisor or kernels of said machine. The network layer in the virtualized environment has inherently unique characteristics, such as providing connectivity between VMs, resource and namespace isolation between tenants, and awareness of migration of VMs. Several designs for virtual networking have been proposed and widely deployed. Popular examples include OpenvSwitch [11], Cisco's Nexus 1000V Switch [12], and VMware's vSwitch [13]. These virtual switching elements behave like a physical switch, but are introduced to interconnect VMs within virtualized or physical networks. We posit that the virtual switch will be a suitable building block for our design for two reasons at least. First, the virtual switch will be a pervasive forwarding plane in commodity hypervisors or even kernels of operating systems. Therefore, our system should be practical and deployable to the existing network devices and systems. Second, it offers rudimentary network isolation for security and performance.

The basic idea of our design is to make a coherent way to dynamically connecting and configuring the virtual switching elements distributed on multiple nodes to enable arbitrary virtual networks. Despite the various implementations, the virtual switching elements have some common designs that are necessary for our system. Basically, a virtual switch contains a set of logical (or virtual) ports, one or more forwarding tables, and some forwarding logics. The logical ports can be bound to virtual interfaces of VMs, physical interfaces, or to other port abstractions such as VLANs or tunnels. Typical examples of the forwarding tables include L2, ARP, or ACL tables. The forwarding logics correspond to the actions for packet processing, notably header rewriting, buffering, filtering, and multicast groupings. We choose the virtual switching elements as the fundamental building blocks of our design, and abstract them as switch, port, table and primitives for forwarding packets.

To support different implementation of virtual switches, our implementation of Virtual Network Management for intra-cloud is logically composed by two major parts: generic control interface and handler. In the following subsections, we explain some important implementation decisions we made to achieve our design goals in the following two subsections.

### B. *Generic Control Interface*

The generic control interface provides a set of functions that execute the operations related to creation and management of virtual networks. These functions serve as a common set of network abstractions to allow users to enable multiple VMs at different sites so they can be interconnected. We implemented a set of functions to control the logical ports, forwarding tables, and forwarding logics. We have abstracted the virtual switching as the following primitives that can describe a virtual network: switching element, virtual port, linking, and policies. Currently, we have implemented the following basic operations to manage a virtual network. However, it is possible to further extend these functions and defines additional primitives.

- Switching element: functions provided for creating, removing, and monitoring a virtual switch in a dedicated node. Since our design shares some common principle of OpenFlow, it does not only dedicate to software switches, but can also be extended to incorporate hardware-based OpenFlow Switches.
- Virtual port: functions provided for adding a virtual port to binding to a VM, removing a virtual port, or disassociating a port with a VM.
- Linking: functions for creating, modifying, and removing connection path between virtual ports. In essence, a virtual network comprises virtual nodes and links that belong to the same experimenter or service provider. We use these functions to create topology for a given virtual network.
- Policy: functions setting a path constraint, such as QoS constraints.

Based on these function libraries, we have implemented some core modules of the virtual network management system. A Virtual Network Description module is implemented to parse the requirement of service requester. After the topology of a virtual network is determined, a VM coordinator module is invoked to communicate with the corresponding nodes to creating virtual switches and bind each virtual port to each VM. Subsequently, a Topology Allocator invokes the functions of control interface to create the linking to reflect the required topology. As a result, multiple VMs within different private cloud systems at different sites can be interconnected, even at sites across the world. Since the assumption and abstraction of the switching elements are generic enough, it can be extended to integrate other orthogonal work on offering an extensible forwarding plane (e.g., [14]) or hardware acceleration (e.g., [15]). We can achieve this extension by implementing a new hander, discussed in the following section.

### C. *Switching Element Handler*

The Virtual Switch Handler serves as the drivers between the required functions of control interface and the underlying virtual switches. The implementations of the Virtual Switch Handlers depend on the implementation of various virtual switches. To prove this concept, we have implemented two handlers: one is for OpenvSwitch and the other is for VMware's vSwitch respectively. The implementation of functions related to supporting the control functions for the switching element and virtual port is straightforward. For examples, we can call the libraries of hypervisor or API of vSwitch to create a virtual switch or a virtual port. In contrast, the implementation of the function to support virtual links is tricky. In general, the virtual link is created to bind two virtual ports if the two virtual ports are connected in the topology. However, we must identify whether the two virtual ports are located in the same LAN. If the two VMs are located across different IP domains without a VPLS mechanism, a GRE [16] tunnel will be created to linking the two virtual ports. If VPLS mechanisms exist in the path, the intra-cloud mechanisms are invoked to create the connectivity. If the two virtual ports belong to the same switch or are located in the same LAN, the VLAN or trunk are used to create the link.

The implementation of OpenvSwitch is designed around OpenFlow [10] so that it exposes a generalized flow table with forwarding actions. As a result, we can utilize the OpenFlow protocol and its API to enable multiple virtual networks on a particular vSwitch by dynamically configuring various VLAN to a particular virtual port or port groups. We can use one single OpenvSwitch in one node to support multiple virtual networks. This approach is subsequently referred to as the VLAN approach in the performance section. Each particular virtual network is populated by dynamically configuring various VLAN in the vSwitch. Nodes configured with the same VLAN can communicate as if they were connected by direct links.

In contrast, VMware's vSwitch does not expose its forwarding plane to be configured outside. Therefore, the handler for VMware's vSwitch creates a different virtual switch in each node for each virtual network. This approach is subsequently referred to as the *vSwitches* approach in the performance section. The major difference is that the implementation of second type requires several virtual switches to enable the same topology as the OpenFlow drivers can do.

## IV. Inter-Cloud Mechanisms

### A. OpenFlow and NetFPGA

In a classical network device (router or switch), the packet forwarding (data path) and the routing/forwarding decisions (control path) occur on the same device. OpenFlow [10] proposes a network architecture that allows decoupling the data plane (the switches) from the control plane (by a controller). The OpenFlow devices forward packets based on flow rules configured by network managers. The controller is in charge of installing the forwarding rules to the OpenFlow devices. We utilize the OpenFlow protocol to control and deploy virtual networks into production networks without disturbing existing isolated traffic. The infrastructure provider can negotiate and delegate virtual segments that will be managed by cloud managers. With the Openflow protocol at the core of our virtual network management system, we can allow cloud managers to define virtual networks by flows and to define the path that they will follow.

NetFPGA [17] is a high-speed, flexible, and hardware-based open platform for network research; it contains four Gigabit Ethernet interfaces and a Xilinx Virtex-II Pro FPGA that can be programmed with user-defined logic. In our system, we implemented our inter-cloud mechanisms in the NetFPGA board, which will serve as the gateway system to convey network traffic between different cloud systems. Multiple high-throughput, customized data planes can be achieved in hardware. The control planes are implemented in POX, a platform for the rapid development of a network control software prototype in Python [18]. POX has the advantage of lower latency and higher throughput as compared to other controller systems. Also, its modular coding style and configuration add much more flexibility for research-purpose developments.

### B. VLAN Translation

We use Fig. 3 as an example to show how the mechanism could operate in a real deployment. Fig. 3 simplified the two virtual networks through two gateway switches to conduct the VLAN translation. The data plane remains, and only processes packets according to the flow table, which is configured by the controller. We extended the controller by implementing two modules and two tables in POX. When a virtual network is deployed, the related information will be updated to two tables in the controller. First, the virtual network management system will determine a path that can connect the VMs of this virtual network. The system will then configure a *vlan2vlan* table of each gateway system along this path. The first module implements the parse daemon for accepting the configured information from the management system and configuring the tables. The *vlan2vlan* table is a tag-to-tag mapping table that enables the private VLAN tag (200 or 100 in our example) be to be recognized and then be translated to the correct public VLAN tag (2782 in our example).

The second table of each gateway system will be updated with the MAC address of the VMs on the foreign network. The second module of the extension on the controller will serve as ARP proxy to respond to the ARP query. Such a design prevents ARP flooding and makes the system scalable and efficient. The module also updates the tables when one VM is migrated or destroyed, since the VM creation, migration, and destruction are controlled by the management system or cloud manager. The management system has full administrative detailed information about the hosts in each network domain. The gateway system manipulates the two tables to do VLAN tag translation between the network domains. With the support of NetFPGA and OpenFlow, the management of VLAN tags translation is maintained in the control plane and leaves the data-plane processing and tag-translation processing in the line rate.

## V. Performance Evaluation

In this section, we describe the system environment, outline our methodology for performance measurement, and present the performance results. We evaluate our system in two ways: with end-to-end performance measurements of a benchmark program on the test environment, and with an application based on the network virtualization mechanism.

### A. Methodology and Environment

It is important that the overhead introduced by the proposed mechanisms be acceptable and not prohibitively high. The performance of virtual networks should be able to handle the applications and the scenarios in which they are used. With the proposed system in this paper, we can easily set up a virtual network as shown in Figure 3 for performance measurement. The purpose of this setup is to create a virtual network with multi-hop paths over real production networks. There are three groups of physical machines, where the first two are located in KUAS center and the third located in another university (NCKU), with eight (8) routing hops between the two universities. The first two groups are allocated in the same LAN but different server rack, and the traffic between them and the third group are across the WAN environment. These machines use 2.50GHz Intel Xeon CPU (E5420) with 4GB RAM and 1GE network interface.

We designed three scenarios to evaluate our implementation. The first scenario connects the VMs with the software bridge that is a native mechanism in kernel or hypervisor for networking. The VMs are connected across the Internet by trunks and VPLS between the distributed switches. This is labeled as the baseline scenario. The second scenario connects VMs by dedicated virtual switches with a GRE virtual link for each virtual network. We label this as the vSwitches scenario. The third scenario connects VMs by the proposed virtual switch with VLAN (port group) mechanism, and is labeled the VLAN scenario in the following subsections We then use some benchmark tools such as BWPing, Nttcp, and Iperf to generate traffic and measure the performance in terms of throughput and latency. The performance of data-bulk services and large transfers depend mostly on throughput, while latency metric is crucial for small transfer and interactive services. The measurements were conducted on each day in the early morning to eliminate interferences to and from production traffic.

## *B. Performance Result*

The throughput performance result is shown in Figure 4. Please note that we only present the result measured from the scenario with GRE Tunnel mechanism, because it serves as the worst case (compared to the scenario with VPLS and bandwidth reserved) in our implementation. The two proposed mechanism performs well in a WAN environment. The optimum throughput achieved is in the range from 800Mbps to 825Mbps. It is interesting that the performance of the vSwitches approach is better than the performance of baseline approach. After further looking into the result, we find the reason for this is because the bridge module is a naïve implementation, just like a software hub. In contrast, the vswitch implementation derived from the OpenvSwitch has some optimization for virtual networking. It is noteworthy that the vSwitches approach is also better than the VLAN approach. The reason is that the vSwitch needs to pay additional overhead on multiplexing or de-multiplexing the packets according to VLAN tag.

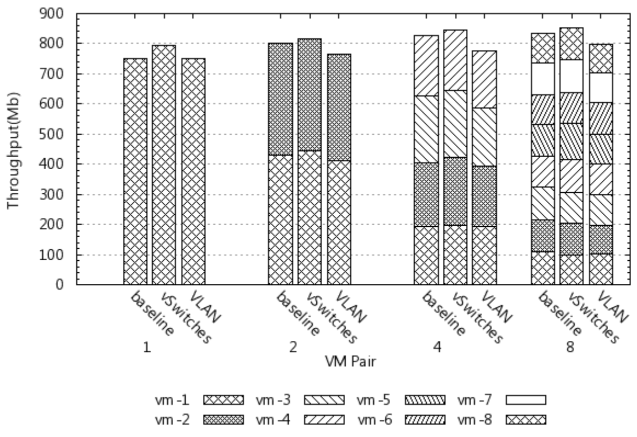

Figure 4: Performance Result of Throughput

To compare latency, we created 32 virtual networks and run a program on each VM pair, in which these programs used the round-trip delay of an ICMP echo request/response pair to measure response time. The measurements were taking in samples over hour-long intervals. We report the average latency in Figure 5. The major additional overhead may come from the GRE tunnel mechanism. As a result, we compare the latency from the vSwitch (labeled with GRE as prefix) and baseline (label with IP as prefix) respectively. The results show that the additional overhead introduced by our approach is acceptable.

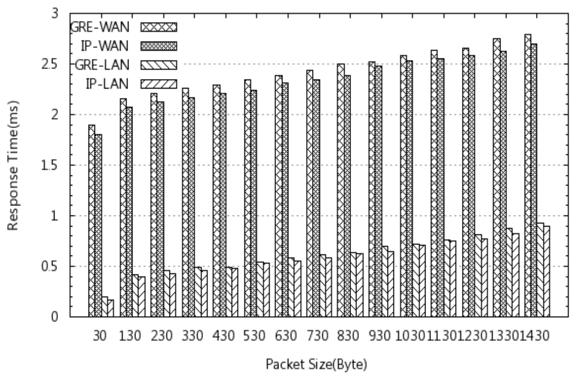

Figure 5: Performance Result of Latency

## *C. Performance of Service Partition*

In our federated model, service can be partitioned on several data centers [5]. Someone may have concern about its performance. Based on the scenario in Figure 1, we conduct a measurement to verify its performance. There are two configurations for this test. The first one (labeled as original) is the original cloud system without service partition. The second one (labeled as federated) is federated cloud with the service partition. We put three benchmark machines in three locations: IDC A (labeled as visiting), IDC B (labeled as Home), and a third place (labeled as neutral). We implemented a special benchmark program to issue session-based requests. The session generator is driven by the scenario defined in Figure 1. We measure the response time of the whole session, as shown in Figure 6. The meaning of the identifiers used in Figure 6 is summarized in Table 1.

TABLE 1

| identifier | corresponding component | Interaction |
|---|---|---|
| login | Login and display the bootstrap page | Portal ⇔ school-db |
| login-check | Authentication | Portal |
| com-post | User profile | School-db ⇔ Portal⇔App Server |
| com-db | Get Content | Portal ⇔ App Server |
| logout | Logout | Portal |
| logout-check | Close session | Portal |

The result shows that the users from the home location can get better performance in federated cloud model. This is because some critical information can be gotten locally. The performance gain is important because most of users are from home location. The results from the neutral location also indicate a better performance in federated cloud model. After tracing the network distance between two IDCs, we found that the reason is the neutral location is near to IDC B. The result implies the location of data center plays an important factor of user perceived latency. The third data set may be the worst-case scenario, in which user locates in the IDC A to access the content from IDC B to IDC A. However, even in the worst-case scenario, our federated cloud model does not have significant performance loss.

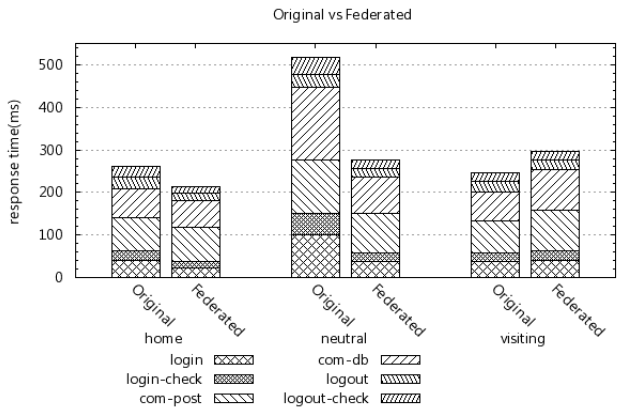

Figure 6: Performance of Service Partition

To support service partition and federated cloud efficiently, the system should make virtual networks easily and dynamically deployable between service instances across production networks. It is important that the additional overhead introduced by creation and forwarding in virtual network should be acceptable and not prohibitively high. The performance of virtual networks should also be able to handle the applications and the scenarios in which they are used. The results in Figure 6 show that the additional overhead introduced by our approach is acceptable.

## VI. Related Work

There is some important research related to enabling the cloud federation. The two representative projects in this direction are the GENI (Global Environment for Network Innovations) [19, 20] in the US and the FIRE (Future Internet Research and Experimentation) [21] in Europe. The common goal of these projects is to build a long-term network environment beyond the capabilities of the current Internet so that researchers can experiment disruptive concepts or systems on a global scale. The network virtualization can play an important role to enable systems with radically different forms of networking within private isolated slices of a shared substrate of these testbeds. However, the exact nature of the control activities for virtualized network, and the design of the control framework, and its implementation are still under active consideration [22]. PlanetLab [23, 24] and ORCA [25] are two of the possible candidates being considered as the control frameworks for GENI and other similar testbeds. However, in contrast to GENI or FIRE that invest a large amount of funding to deploy another national-level fiber for researches, our system was built upon the existed production infrastructures. With the federated cloud with virtual network, we are currently trying to provide a *network experiments as service* for researchers in Taiwan.

Some research efforts have tried to complement this and improve the capability of control on virtual networks. VINI [26] is a testbed platform that extends PlanetLab by introducing the concept of network virtualization. In VINI, routers are virtualized by User Mode Linux and interconnected by virtual links. As such, VINI allows researchers to deploy experiments with arbitrary network topologies simultaneously. VIOLIN [27] (Virtual Internetworking on OverLay INfrastructure) is an application-level virtual network architecture, where mutually isolated virtual networks can be created on top of an overlay infrastructure such as PlanetLab. Although these platforms can run multiple networking experiments in parallel, forwarding packets in user space significantly limits scalability. An updated VINI platform, Trellis [28], allows for higher forwarding performance by introducing container based virtualization techniques for both system and network stack virtualization. Our work makes coherent network virtualization in the hypervisor and with the support of NetFPGA, causing a good performance and isolation.

The above mentioned systems mainly build on the architecture and management framework of PlanetLab, however, they are not integrated with existing services over wide-area production infrastructure that reflect a real network deployment.

## VII. Conclusion

This paper summarizes our experience in designing and implementing an integrated system to support network virtualization for federated cloud systems. Unlike state-of-the-art solutions, the presented solution can provide dynamic virtual networks for a federation of independent infrastructure providers across the production networks. In the context of cloud computing, our work enables the creation of federated cloud networks. The multi-layer, dynamic provisioning potentials, and distributed nature of this system presents interesting challenges and also major opportunities for addressing many types of research topics. There are many different unexplored areas for further research. First, efficient allocation and scheduling of physical resources among different virtual networks to maximize the utilization of the system resources is not well addressed in this paper. Second, the negotiation and enforcement of service level agreement across multiple administrative domains is needed for some critical applications or enterprise services. Third, we are exploring protocols and solutions to deal with inter-domain network administration issues.

## Acknowledgements

This research was financially supported by the National Science Council of Taiwan, under grants No. NSC 100-2218-E-151 -003 -MY3 and NSC 101-2622-E-151 -026 -CC3. The authors also thank Pan-Wei Tsai from NCKU, and Te-Lung Liu from NCHC for their support on network facilities and WAN configurations.

## REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," ACM SIGCOMM Computer Communication Review, vol. 39, pp. 50-55, 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "A view of cloud computing," Communications of the ACM, vol. 53, pp. 50-58, 2010.

[3] C. J. S. Decusatis, A. Carranza, and C. M. Decusatis, "Communication within clouds: open standards and proprietary protocols for data center networking," Communications Magazine, IEEE, vol. 50, pp. 26-33, 2012.

[4] M.-Y. Luo, "Design and Implementation of an Education Cloud," in Journal of Internet Technology.

[5] M.-Y. Luo and S.-W. Lin, "From Monolithic Systems to a Federated E-Learning Cloud System," in IEEE International Conference on Cloud Engineering, San Francisco, California, USA, 2013.

[6] iGENI: A Distributed Network Research Infrastructure for the Global Environment for Network Innovation http://groups.geni.net/geni/wiki/IGENI

[7] M. Lasserre and V. Kompella, "Virtual private LAN service (VPLS) using label distribution protocol (LDP) signaling," RFC 4762, January 2007.

[8] C. Kim and J. Rexford, "Revisiting Ethernet: Plug-and-play made scalable and efficient," in Local & Metropolitan Area Networks, 2007. LANMAN 2007. 15th IEEE Workshop on, 2007, pp. 163-169.

[9] A. Myers, E. Ng, and H. Zhang, "Rethinking the service model: Scaling Ethernet to a million nodes," in Proc. ACM SIGCOMM Workshop on Hot Topics in Networking, 2004.

[10] 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, vol. 38, pp. 69-74, 2008.

[11] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," Proc. of Workshop on Hot Topics in Networks (HotNets-VIII), 2009.

[12] Cisco nexus 1000v series switches. http://www.cisco.com/en/US/products/ps9902.

[13] Vmware vnetwork distributed switch. http://www.cisco.com/en/US/products/ps9902.

[14] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: exploiting parallelism to scale software routers," in Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, 2009, pp. 15-28.

[15] PCI-SIG: PCI-SIG Single Root I/O Virtualization and Sharing Specifications 1.1. http://www.pcisig.com/specifications/iov/single_root/.

[16] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. GenericRouting Encapsulation (GRE). Internet Engineering Task Force, March 2000. RFC 2784.

[17] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA--an open platform for gigabit-rate network switching and routing," in Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on, 2007, pp. 160-161.

[18] POX controller. Available: http://www.noxrepo.org/pox/about-pox/

[19] GENI: Global Environment for Network Innovations, http://www.geni.net

[20] G. P. Group, “GENI design principles,” Computer, vol. 39, no. 9, pp. 102-105, 2006.

[21] FIRE: Future Internet Research and Experimentation, http://cordis.europa.eu/fp7/ict/fire/

[22] Subharthi Paul, Jianli Pan, and Raj Jain, "Architectures for the Future Networks and the Next Generation Internet: A Survey," Computer Communications, UK, Volume 34, Issue 1, 15 January 2011, Pages 2-42.

[23] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak, “Operating system support for planetary-scale network services,” In NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation, Berkeley, CA, USA, 2004.

[24] N. Spring, L. Peterson, A. Bavier, and V. Pai, “Using PlanetLab for network research: Myths, realities, and best practices,” SIGOPS Operating Systems Review, vol. 40, no. 1, pp. 17-24, 2006.

[25] Jeff Chase, Ionut Constandache, Azbayer Demberel, Laura Grit, Varun Marupadi, Matt Sayler, and Aydan Yumerefendi. Controlling dynamic guests in a virtual computing utility. In International Conference on the Virtual Computing Initiative (ICVCI 2008), May 2008.

[26] Andy C. Bavier, Nick Feamster, Mark Huang, Larry L. Peterson and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In Proceedings of the SIGCOMM Conference, pages 3-14, Piza, Italy, September 2006.

[27] Xuxian Jiang and Dongyan Xu. Violin: Virtual internetworking on overlay infrastructure. In International Symposium on Parallel and Distributed Processing and Applications (ISPA) 2004, 2004.

[28] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. “Trellis: A platform for building flexible, fast virtual networks on commodity hardware”. Proc. Workshop on Real Overlays and Distributed Systems, 2008.