Enabling SDN Applications on Software-Defined Infrastructure

Thomas Lin, Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada
Email: {t.lin, joonmyung.kang, hadi.bannazadeh, alberto.leongarcia}@utoronto.ca

***Abstract*—In this paper we discuss how to enable Software-Defined Networking (SDN) applications on Software-Defined Infrastructure (SDI) which is an approach for integrated control and management of converged computing and networking resources. Current separated resource management for computing or networking resources is not sufficient for addressing applications and multimedia services that require guaranteed service and quality levels. In addition, current resource management is not capable of managing heterogeneous resources that include computing and networking resources in combination with other resources such as programmable hardware, GPUs and network processors. We present an SDI that provides pluggable resource management modules for scheduling, networking control, fault management, and so on. This paper focuses on the design and implementation of a networking control module that enables SDN applications using information available from other modules. Currently, we have deployed the network control module in the practical multi-tier cloud infrastructure, SAVI Testbed. We present real measurements that show the functional evaluation results of our network control module.**

I. Introduction

In cloud computing, a cloud controller is responsible for taking the high-level user descriptions, and then managing computing resources, placing virtual machines (VMs), allocating storage, deciding where the image will run, and attaching networking to meet resource needs [1]. Software-defined networking (SDN) is an approach to building data networking equipment and software that separates and abstracts elements of these systems [2]. SDN allows system administrators to provide network services more easily through abstraction of lower-level functionality into virtual services. An SDN controller is an application that manages flows to enable more flexible, customized, and intelligent networking. SDN controllers are based on protocols, such as OpenFlow [3], that allow servers to configure switches on how to process packets and where to forward them.

Current approaches that use two separated resource management systems (one for computing and one for networking) are not sufficiently capable and flexible to address applications and multimedia services that require guaranteed service and quality levels. The end-to-end quality of a service or application is determined by the performance of underlying computing and networking resources, and so these resources must generally be managed in coordinated fashion. Accordingly, approaches that separate resource management of cloud or network resources are not able to provide guarantees. Integrated management of computing and networking resources enables new management capabilities. For example, unlike previous approaches, integrated management of converged resources can provide energy-aware forwarding and resource allocation because energy consumption information for the computing cloud and the network are shared.

978-1-4799-0913-1/14/$31.00 © 2014 IEEE

While computing and networking resources provide the bulk of the support for cloud-based applications, other resources such as programmable hardware, GPUs, and network processors provide critical support for certain services and applications. Current resource management systems are not capable of managing heterogeneous resources that include computing and networking resources in combination with other resources. In addition, current management systems do not apply virtualization methods to heterogeneous resources, and therefore are not capable of realizing the flexibility, scalability and economic advantages that would be inherent in the integrated management of converged heterogeneous resources.

To overcome shortcomings of current management approaches, the Smart Applications on Virtual Infrastructure (SAVI) project has proposed Software Defined Infrastructure (SDI) as an approach for integrated control and management of converged heterogeneous computing and networking resources in *software* [4], [5]. In SDI, a centralized SDI manager controls both computing and networking resource management through a cloud controller and an SDN controller. SDI has pluggable resource management modules for scheduling, networking control, fault management, and etc. In this paper, we focus on a networking control module to enable SDN applications on SDI by defining north- and south-bound APIs between an SDI manager and an SDN controller. Currently, we have deployed the network control module in the practical cloud infrastructure, SAVI Testbed, and demonstrated how to manage not only physical networking but also virtual networking based on the deployed system [6]. We show real measurement data for the running networking control module in the SAVI Testbed by increasing the number of virtual machine instances or control modules.

The paper is organized as follows. Section II describes related work from SDN, OpenFlow, FlowVisor, as well as the integration of cloud and SDN controllers. Section III presents a high-level architecture and design of an SDI resource management system. Design and implementation of a network control module in SDI are described in Section IV. In Section V, we present example SDN applications that run on the SDI networking control module. Functional evaluation results of our proposed SDI network control module are shown in Section VI. Finally, conclusions and future work are presented in Section VII.

II. Related Work

Recently, a tremendous amount of research has addressed cloud data centers and software-defined networking (SDN). Only a few efforts have addressed the integration of cloud computing and SDN, namely in terms of Network-as-a-Service (NaaS), and integrated management as in SDI.

Benson et al. proposed CloudNaaS as a cloud networking platform for enterprise applications [7] in which a networking framework extends the self-service provisioning model of the cloud beyond virtual servers and storage to include a rich set of accompanying network services. CloudNaaS allows customers deploying their applications on the cloud to access virtual network functions such as network isolation, custom addressing, service differentiation, and the ability to deploy middlebox appliances to provide intrusion detection, caching, or application acceleration.

OpenFlow is a leading mechanisms for providing SDN in cloud datacenters [3]. OpenFlow controllers such as NOX [8], POX [9], Ryu [10], or Floodlight [11] are available and written in different programming languages. Previous research have investigated the performance and scalability of the controllers [12], [13], [14], [15]. Currently, we are using the Ryu OpenFlow controller in our SAVI Testbed [5] as an SDN controller.

In order to manage networking resources using OpenFlow in a cloud datacenter, Neutron (formerly Quantum in the OpenStack project) can be used to provide NaaS between interface devices managed by other OpenStack services. One of the main features in Neutron is to enable innovation plugins (open and closed source) that introduce advanced network capabilities for OpenFlow controllers. Recently we have implemented an SDI plugin for Neutron to fully integrate the SDI manager in OpenStack.

Sherwood et al. proposed a FlowVisor to enable switch virtualization so that the same hardware forwarding plane can be shared among multiple logical networks, each with distinct forwarding logic [16]. Like a hypervisor for system virtualization, the FlowVisor uses OpenFlow as a hardware abstraction layer to sit logically between the control and forwarding paths on a network device for network virtualization. In our current system, the SDI manager controls a FlowVisor for creating slices, assigning virtual networks to the slices, or deleting slices.

III. Overview of SDI Resource Management

In this section, we present a system architecture and design of an SDI Resource Management System (RMS) for converged heterogeneous resources.

A. System Architecture

Figure 1 shows a high-level architecture of the SDI RMS, in which an SDI manager can control and manage the resources using a cloud controller, a network controller, and a topology manager. External entities obtain virtual resources in the converged heterogeneous resources via the SDI RMS. The converged heterogeneous resources are composed of virtual resources and physical resources. Virtual resources include any resource virtualized on the physical resources such as virtual machines. Physical resources include any resource that can be abstracted or virtualized, such as computing servers, storage, and network resources (routers or switches). The SDI RMS provides general resource management functions for the converged heterogeneous resources to the external entities. The resource management functions include provisioning, registry/configuration management, virtualization, allocation/scheduling, migration/scaling, monitoring/measurement, load balancing, energy management, fault management, performance management (delay, loss, etc.), and security management (authentication, policy, role, etc.). The external entities can be applications, users (service developers or providers), and high-level management systems.

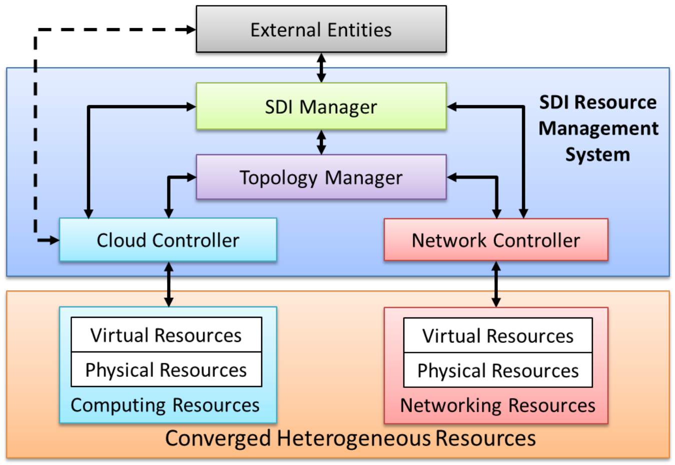

Fig. 1. High-level architecture of resource management system for SDI

We envision the SDI manager as a way to realize major integrated resource management functions: fault tolerance, green networking (energy efficient and/or low-carbon emitting), path optimization, resource scheduling optimization, network-aware VM replacement, QoS support, real-time network monitoring, and flexible diagnostics based on network topology information from a topology manager. The cloud controller is responsible for taking the high-level user descriptions and managing computing resources, placing virtual machines, and allocating storage. The SDN controller takes a network specification and translates it into high level configuration commands that can be installed on SDN-enabled networking resources. The topology manager maintains a list of the resources, their relationships, and monitoring and measurement data of each resource. Furthermore, the topology manager provides up-to-date resource information to the SDI manager for topology-aware resource management. The SDI manager uses the cloud controller for computing resource provisioning, migration, load balancing, and scaling, while the cloud controller provides the requested virtual computing resources to the SDI manager. Similarly, the SDI manager uses the SDN controller for controlling and managing networking resources, and the SDN controller provides virtual network resources and monitoring data to the SDI manager in return. The SDI manager uses the topology manager for setting resource cost properties and metrics, as well as updating resource data, whereas the topology manager provides physical and virtual network topology and associated status information, as well as resource monitoring and mea-

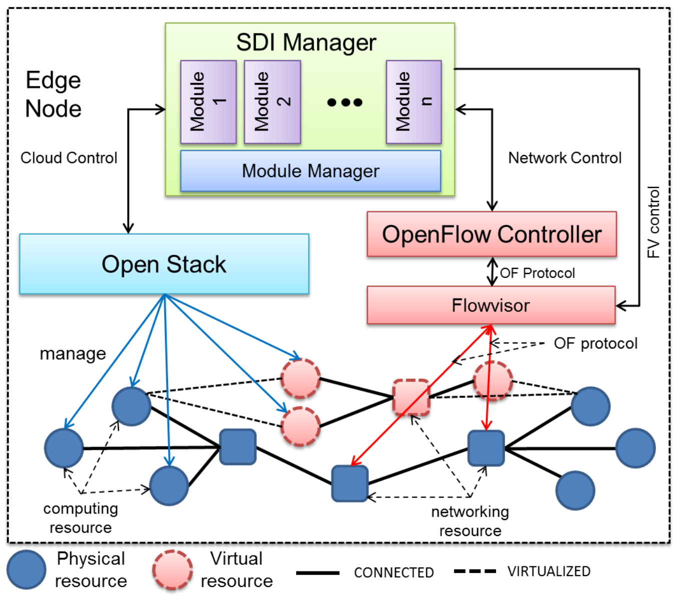

Fig. 2. High-level design of SDI resource management system using OpenStack and OpenFlow controller

surement data to the SDI manager.

B. Design of SDI Control and Management System

Figure 2 shows that the design of a SAVI Smart Edge node based on the SDI architecture includes four majors parts: 1) Edge node network, 2) OpenStack, 3) OpenFlow controller, and 4) SDI manager [4]. In the Edge node network, a variety of heterogeneous computing and networking resources are available.

As shown in Figure 2, the SDI manager controls and manages virtual computing resources by virtualizing physical computing resources using OpenStack [17]. The OpenFlow controller [3] is used for controlling networking resources. The OpenFlow controller receives all events from the OpenFlow switches and creates a flow table including actions. The SDI manager performs all management functions based on the data from the OpenStack and the OpenFlow controller, and determines appropriate actions for computing and networking resources. The SDI manager has a module manager to manage specific functional modules such as a scheduling module, a networking control module, a fault tolerant management module, or a green networking module. In this paper we focus on the networking control module which is responsible for enabling SDN applications, which we will discuss in detail in Section 4. Details of the other modules are out of the scope of this paper. The OpenFlow controller may include a proxy that mediates access from multiple OpenFlow controllers to the networking resources. In our system, a FlowVisor [16] acts as a transparent proxy between the OpenFlow switches and multiple OpenFlow controllers. The FlowVisor creates slices of network resources and delegates control of each slice to a different controller, while enforcing isolation between the slices. The introduction of FlowVisor enables any user to use their own OpenFlow controller, even though it may be outside the system. Internally, we have used the Ryu OpenFlow

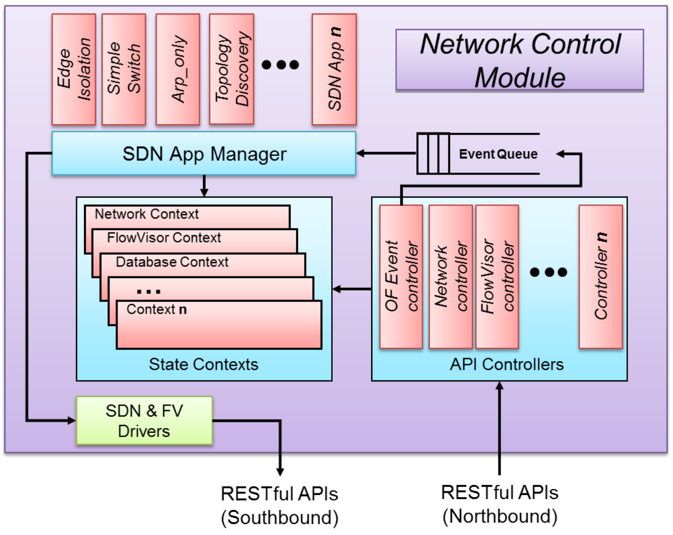

Fig. 3. Design of network control module in SDI

controller [10]. Via FlowVisor, any user can then access and control his or her own slice of the network.

As in SDN, we have separated the data and control planes in the Smart Edge. The OpenStack and OpenFlow controller are modules for communicating directly with computing and networking resources, whereas the SDI manager in Figure 2 of the Smart Edge is responsible for C&M tasks.

IV. Network Control Module in SDI Manager

In this section, we will describe the design and current implementation of the network control module running atop of our SDI manager.

A. Design of Network Control Module

Figure 3 shows a high level architectural view of the network control module. Essentially, the module enables the ability to run one or more network control applications (e.g. Learning Switch, Topology Discovery, etc.). These applications define the behaviour of the network and are able to interact with the network elements below via a programmatic interface, much like regular network operating system applications. However, the implementation of the backend for handling the interaction with the network varies depending on the SDN controller being utilized below the SDI manager.

The network control module defines a set of Representational State Transfer (RESTful) APIs [18] that can be used by the SDN controller and other external clients, such as the Neutron OpenStack component [19] for reporting changes in the network configuration. As seen in Figure 3, northbound APIs are assigned to one of many backend controllers for processing. New APIs and corresponding controllers can be easily added to expand the functionality supported by the network control module. For example, the current version of the network control module supports port bonding and the use of FlowVisor, both of which are configurable via RESTful APIs.

Since the proper execution of a network control application may depend on the current configuration of the network, the network control module saves any configuration settings that may be needed in different contexts (see Figure 3). The configuration is then used by the network control applications to provide a context for the decisions it must make. As this information may need to be accessed on a per-packet basis, it is kept in memory rather than in a remote database in order to reduce the access latency. To protect against losing the entire configuration in the event of a system crash, any configurations that cannot be re-learned on-the-fly (e.g. ports on a switch to be bonded) are pushed to an SQL database (not shown in the Figure 3) as a precaution. In the event that the SDI manager crashes, the configuration settings can be immediately reloaded upon start-up of the network module in order to recover its previous state.

B. Northbound APIs

We define the set of northbound APIs to be calls that originate from some external entity and are received by the network control module. These APIs enable the module to receive notifications of changes in the network configuration from external clients as well as event messages from the SDN controller (e.g. packet-in event). Network configuration changes may include, but are not limited to, events such as the creation/deletion of new virtual networks, registration/removal of ports in the system, the migration of an interface from one virtual network to another, the delegation of control over a virtual network, and etc. As these notifications affect the configuration of the network, they are handled synchronously by the SDI manager in order to ensure that future packets are processed with respect to the most up-to-date network configuration available. Conversely, packet-in events, which arrive much more frequently and may take some time to process, are inserted into a queue and handled asynchronously by the system in order to prevent delaying (or possibly blocking) the receipt of configuration change notifications.

C. Southbound APIs

The southbound APIs are defined as those that originate from the network control module and are received by the SDN controllers running below the SDI manager. Once a network control application reaches a verdict on how to handle a packet, it must contact the SDN controller in order to execute its decision. These southbound APIs are controller-dependent and thus vary in their implementation. As different controllers may have different APIs, the SDI manager maintains a set of drivers for each type of controller. In our current implementation, these APIs are used to inform the controller on how to handle a packet (e.g. drop, flood, or output a packet, etc.) as well as to configure the flow table in switches. The benefit of this design is that it does not restrict the SDI manager to using any one controller, and opens up the possibility of using any future SDN controllers that may use non-OpenFlow protocols to communicate with the network fabric. In essence, from the viewpoint of the SDI manager, the SDN controller is merely a network interface layer.

Another important set of southbound APIs initiated from the SDI manager are those used to configure and control FlowVisor. FlowVisor comes with a default set of RESTful APIs, using JSON-RPC, that can be used by any external client. Like the other drivers used to communicate with the SDN controllers, the SDI manager also has a separate driver specifically for communicating with FlowVisor using its established APIs.

D. OpenStack, Ryu, and FlowVisor

The current deployment of the SDI manager within the SAVI testbed [5] integrates it with OpenStack, and uses the Ryu OpenFlow controller [10] as the primary SDN controller.

1) Interaction with OpenStack: Neutron, the networking component of OpenStack, is a system that aims to provide “Networking as a service” [19]. Neutrons responsibility is to keep all the necessary network-related information (e.g. MACs, attached interfaces, etc.) of the virtual computing resources located throughout the cloud. For the task of actually controlling the network, Neutron delegates this responsibility to a number of plugins (i.e. backend controllers). We use the network control module APIs to allow Neutron to relay the networks configuration information to the SDI manager. Originally, we modified the Ryu plugin client to act as a proxy to forward all the configuration change notifications regarding virtual networks and virtual interfaces to the SDI manager. We have recently developed a plugin specifically for our SDI manager and thus have fully integrated it with Neutron. Additionally, we have extended the set of information that Neutron reports to its plugin to also include information regarding the assigned MAC addresses and, where it is applicable, the IP addresses of interfaces. With this information, the SDI manager has the networking-related context to properly control and manage the communications and network connectivity between the various computing resources within the cloud.

2) Interaction with the SDN Controller: While any SDN controller can be used below the SDI manager, it is important that whichever controller is chosen is able to perform certain tasks. In particular, the SDN controller must support:

1) RESTful web services to receive and process requests from the SDI manager. Through these APIs, the SDN controller serves as an OpenFlow interface layer for the SDI manager and any other components wishing to interface with the network switches. Examples of currently available APIs in our deployment include the ability to write and delete flows into switches, send custom packets from switches, query switch statistics, and query the latest network topology;
2) An OpenFlow event forwarding application that forwards event notifications from network switches to the SDI manager. As the network control module is responsible for handling the packet routing decisions for the network, the SDN controller must forward the relevant information to the SDI manager. The application in use in our SAVI Smart Edges require only the input port, datapath ID, source, and destination MACs in order to make a routing decision. Thus, upon receipt of a packet-in event, the forwarding application must parse the packet for the necessary header fields and send this information northbound to the SDI manager via a RESTful request.

In our deployment, the above two tasks are performed by applications running on top of Ryu. An optional, third application that we have running is a topology discovery application that periodically sends Link Layer Discovery Protocol (LLDP) packets through the network to discover the various active links in the system. The resulting topology can then be queried via a RESTful API. While we can also run a topology discovery application on the network control module itself, we prefer to delegate tasks that do not benefit from a centralized view of the network to the lower layers to avoid the overhead of extra API calls going through multiple components.

3) Interaction with FlowVisor: The addition of FlowVisor into the SAVI testbed allows users the option of delegating control of one or more virtual networks to another OpenFlow controller running outside the system. Since the SDI manager has a complete view of the network, including the MAC addresses of computing resources interfaces as well as which ports in the network they are connected to, it is well positioned to make decisions on how to slice the network into separate FlowSpaces. Via the FlowVisor driver, the SDI manager is able to install a series of FlowSpace entries that effectively delegates control over a subset of the underlying network topology to a users guest controller. Currently, the FlowSpace definitions are based on three parameters, that of the datapath ID, the port number, and the MAC address. From this information, we slice the network at L2, thus granting users the freedom to experiment with novel networking protocols pursuant of the goal to support Future Internet research on the SAVI testbed.

If no networks have been delegated, FlowVisor has a default admin slice with a corresponding default rule matching all flows and set at the lowest priority level. This default rule forwards all packets to the main SDN controller, which in turn forwards the relevant packet information up to the SDI manager.

V. SDN Applications

Since the network control module stores the network configuration, it is able to provide the proper context from which the networking application(s) bases its decisions on. All the SAVI Smart Edges currently run the same network control application, which is primarily designed to ensure the proper isolation of traffic between each virtual network as defined by Neutron. In this section, we will briefly describe the logic of our isolation mechanism as well as the other port bonding and FlowVisor features currently supported in our application.

A. Virtual Network Isolation

The SAVI Edge application currently used to ensure the proper isolation of virtual networks was developed on top of a base application that was originally written by the creators of Ryu. The application utilizes two key pieces of information provided by Neutron: virtual network IDs, and a mapping of each switch port throughout the network infrastructure to one of these network IDs. With this information, the application is able to determine which ports on a switch are allowed to communicate with each other. Ports connected to the interfaces of computing resources are always registered with the ID of the virtual network to which they belong. The application also defined a special-case "external" network ID, used to

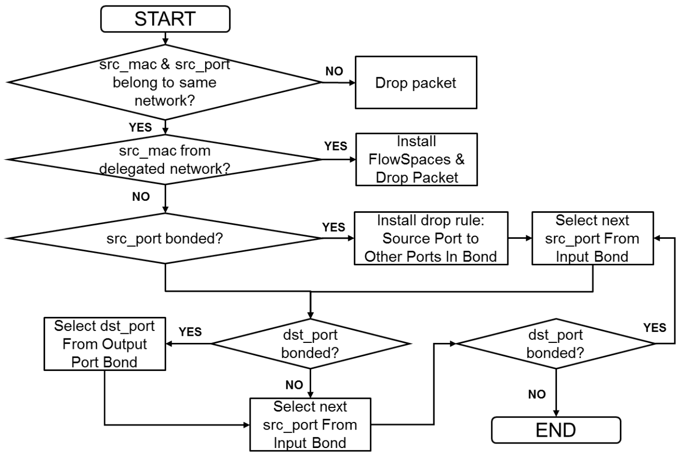
START
src_mac & src_port belong to same network?
NO
Drop packet
YES
src_mac from delegated network?
YES
Install FlowSpaces & Drop Packet
NO
src_port bonded?
YES
Install drop rule: Source Port to Other Ports In Bond
Select next src_port From Input Bond
Select dst_port From Output Port Bond
YES
dst_port bonded?
NO
Select next src_port From Input Bond
dst_port bonded?
YES
NO
END

Fig. 4. A Flowchart of Packet Handling Logic

signify that any port registered to this external network can allow traffic from any virtual network to pass through it. Such a special-case ID is needed due to inter-switch links, which must support traffic from multiple tenants resources. When a packet arrives through one of these external ports, the network it belongs to is determined by the source MAC address of the packet.

As previously mentioned, we have modified Neutron to call the network module APIs in order to register the MAC-to-network ID associations. This explicit registration is important as it is used in two of the mechanisms we have in place to prevent potential MAC address spoofing, which malicious users may attempt use to bypass the network isolation. One mechanism to prevent spoofing is a simple check to see if the network ID associated with the source MAC address is identical to the network ID associated with the port it is coming from. Similarly, checks are made to ensure the network IDs associated with the source and destination MACs are identical as well. With the network configuration information regarding the MAC and port associations with virtual network IDs, the packet handler has enough context to properly process incoming packets. The application essentially operates as a per-virtual network learning switch.

B. Packet Handling Logic for Port Bonding and FlowVisor

As the network control module includes support for port bonding and FlowVisor, our application must take this information into consideration when processing packets. The ports contained within a bond are pre-vetted at registration-time to ensure they belong to the same virtual network. As shown in Figure 4, if a packet arrives from a bonded port, rules are automatically installed into the switch to ensure that the other ports within the same bond cannot be used as the output port (i.e. install pre-emptive drop rule). Additionally, if the application determines that a flow rule to be installed where the input port belongs to bond, additional flow rules will be installed, with modified input port match fields, for each port within that bond. Similarly, if the output port of a flow is bonded, its actual output port for the flow will be chosen in round-robin fashion over the ports within the bond.

Occasionally, if a problem occurs in FlowVisor that causes

FlowSpace rules to be deleted, a packet that was destined for a guest OpenFlow controller will instead go to the default SDN controller, which forwards it to the SDI network control module to be handled. To cover such a case, the application always checks if the network ID associated with a packets source MAC has been delegated to a guest controller (See Figure 4). If the network has been delegated, the application first re-installs the appropriate FlowSpace rules if necessary, then drops the packet, so that future packets will be routed to the proper controller.

VI. Evaluation

In this section, we perform a functional evaluation of our network control module in SDI using the SAVI Testbed [5]. We first describe the experimental setup used to carry out the evaluation, and we then present our experimental results.

A. Experimental Setup

We run the SDI manager in a server which has two Intel Xeon E5-2650 CPUs with a total of 16 cores (32 hyperthreads), clocked at 2.0 GHz per core, and 64 GB of system RAM. The SDI manager itself is fully implemented using Python version 2.7.

To demonstrate the feasibility of the system, we designed an experiment in attempt to discern the throughput and scalability of the SAVI network application. After pre-loading the queue with packets (see Figure 3), we start the application and observe the throughput based on the queue consumption rate. This experiment is repeated with increasing number of instances of the control application.

The experiment uses a dedicated server on a minimal installation of Ubuntu 12.04, in order to ensure a clean evaluation of the control application. Since the experiment aims to measure the maximum throughput of the SAVI network application, this represents the worst-case scenario where the SDI manager must do packet-by-packet processing. In the network management of the SAVI Smart Edges, this will not be the case as new packets will trigger flow table entries to be installed into the OpenFlow switches.

B. Experimental Results

We report on the throughput of the network control module running the SAVI network application. In order to observe the scalability of our network control application, we run it in a dedicated server. Figure 5 shows the total throughput of packet-in requests through the system with the pre-loaded queue against increasing number of control modules running our SAVI network application. The total throughput is a sum of each control modules throughput. The results show promising results regarding the scalability of the network control module atop the SDI manager, as the total throughput is near-linear in the number of control module instances. It means that the network control module can handle growing amounts of packet-in requests in a graceful manner.

It is worth noting that the system presented herein is currently used in all the SAVI nodes controlling the production network. As of September 2013, the SAVI testbed [6] comprises one Core node and six Edge nodes, all of which are available for providing resources to run applications or experiments on. In order to show the current system is able to handle the network load of the testbed, we picked the Core node because it contains many active virtual machines and running experiments. The Core node consists of one controller, eleven interconnected computing servers (each running an Open vSwitch [20] within), one object storage server, one volume server, and two physical OpenFlow-enabled switches. At the time of collecting the network load data of the Core node, there were fourteen projects, fifty users, and about one hundred and fifty virtual machines and other computing resources for active experiments owned by SAVI researchers. In terms of the load on the networking control module in the SDI manager, we measured the number of packets received by the network control module using *dumpcap*, which is a command line traffic monitoring tool. Figure 6 shows the packet-in requests per hour over roughly four weeks on the node. The results show that our network control module is able to completely processes any amount of requests from all running virtual machines and computing resources in the SAVI testbed in a steady manner. As a result, we believe that this demonstrates the practicality of our network control module to enable SDN applications on real multi-tier cloud datacenters.

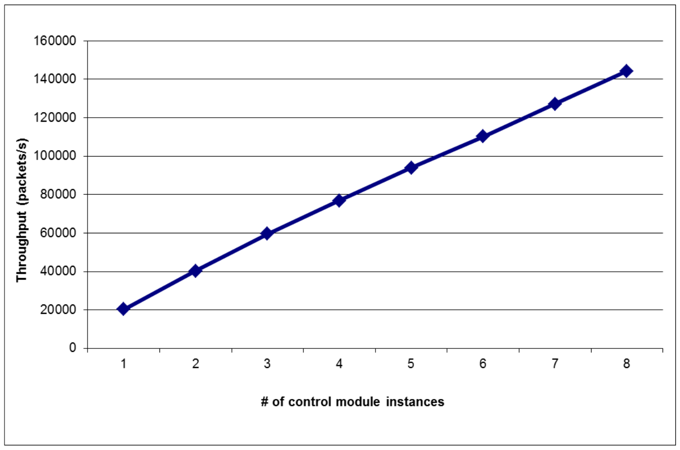

Fig. 5. Throughput of packet-in requests vs Increasing the number of network control module instances

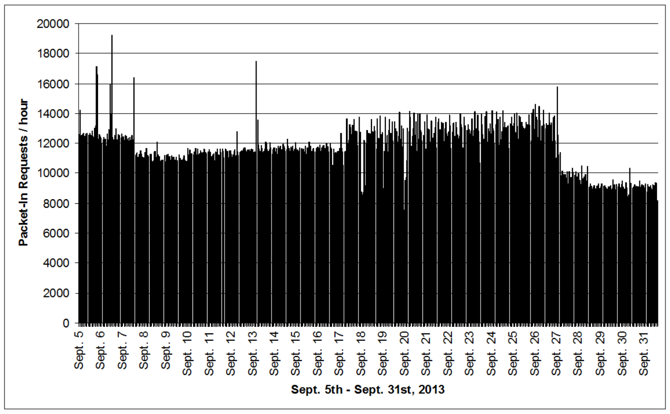

Fig. 6. Hourly packet-in requests to the SDI manager of the largest node in SAVI testbed

VII. Conclusions

In this paper we have presented an SDI resource management system that provides integrated control and management for converged heterogeneous resources using a cloud controller and an SDN controller. We have proposed how to enable SDN applications using a networking control module in the SDI manager, as well as defined the southbound and northbound APIs for communication between the SDI manager and SDN controller. Some sample SDN applications, such as virtual network isolation and the packet handling logic for port bonding and FlowVisor, were discussed as well. We have shown that the throughput of our network control module in SDI is scalable by increasing the number of control module instances, thus demonstrating its capacity for controlling and managing the network load of an applications testbed. Currently, we have deployed the network control module in the real practical data center, the SAVI testbed and shown that it has been working well. As the network control module is constantly aware of both computing and networking resources within the testbed, we believe this work presents a concrete first step towards realizing more complex infrastructure-aware network management schemes. In future work, we will improve the performance and scalability of the SDI manager including the network control module. We will also implement more SDN applications for showing practicality and continue to monitor and measure data to show the long-term stability of the system.

Acknowledgment

The work for all the published papers which follow is funded in part or completely by the Smart Applications on Virtual Infrastructure (SAVI) project funded under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP394424-10.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] N. McKeown, "Software-defined networking," SIGCOMM 2009, Keynote, 2009.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] J.-M. Kang, H. Bannazadeh, H. Rahimi, T. Lin, M. Faraji, and A. Leon-Garcia, "Software-Defined Infrastructure and the Future Central Office," in *IEEE International Conference on Communications, IEEE ICC13 - CNDC*, Budapest, Hungary, June 9-13 2013.

[5] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "SAVI testbed: Control and management of converged virtual ICT resources," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, Ghent, Belgium, 2013, pp. 664–667.

[6] J.-M. Kang, T. Lin, H. Rahimi, M. Faraji, H. Bannazadeh, and A. Leon-Garcia, "2013 SAVI Testbed Workshop," http://www.savinetwork.ca/news-events/savi-testbed-workshop-agm-july-4-5-2013/.

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

[8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[9] J. Mccauley, "POX: A Python-based OpenFlow Controller," http://www.noxrepo.org/pox/about-pox/. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[10] F. Tomonori, "Introduction to ryu sdn framework," Open Networking Summit, April 2013, http://osrg.github.io/ryu/slides/ONS2013-april-ryu-intro.pdf.

[11] B. S. Networks, "Floodlight," http://www.projectfloodlight.org/floodlight/.

[12] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *Communications (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.

[13] M. Fernandez, "Evaluating OpenFlow Controller Paradigms," in *ICN 2013, The Twelfth International Conference on Networks*, 2013, pp. 151–157.

[14] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012.

[15] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[16] R. Sherwood, G. Gibby, K.-K. Yapy, G. Appenzellery, M. Casado, N. McKeown, and G. Parulkary, "Flowvisor: A network virtualization layer," OpenFlow, Tech. Rep. OPENFLOW-TR-2009-1, October 2009.

[17] "Openstack," http://www.openstack.org.

[18] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[19] OpenStack, "Neutron openstack project," http://docs.openstack.org/developer/neutron/.

[20] Nicira Networks, "Open vSwitch: An open virtual switch," http://openvswitch.org/.