Heterogeneous Interconnection between SDN and Layer2 Networks based on NSI

Ta-Yuan Chou, Wun-Yuan Huang, Hui-Lan Lee, Te-Lung Liu, Joe Mambretti*, Jim Hao Chen*, Fei Yeh*
National Center for High-Performance Computing, NARLabs, Taiwan
*International Center for Advanced Internet Research (iCAIR), Northwestern University, U.S.A
{ 1203053, wunyuan, gracelee, tlliu}@narlabs.org.tw
{j-mambretti, jim-chen, fyeh}@northwestern.edu

***Abstract*—Recently, Software Defined Network (SDN) technology has become widely used to take advantage of its features, including capabilities for optimization, flexibility, and customization. Many companies and organizations are applying this technology to build SDN networks for testbeds or production networks to address their requirements for optimization and customization. Currently, north bound and south bound APIs, which integrate SDN controllers and switches, are widely defined and implemented. However, an east-west interface standard for the communications among several SDN domains does not yet exist. In this paper, we propose such a method using Network Service Interface (NSI) to solve the east-west communication problem among SDN domains. The proposed method first uses OpenVirtex as the hypervisor in each SDN domain. We also extend OpenNSA, an open-source implementation of NSI agent, to deal with local OpenVirtex networks for communicating with other SDN domains. In addition, by using the standard NSI protocol, cross-platform interconnections between SDN and legacy networks can be achieved.**

Keywords—SDN; NSI; OpenNSA; OpenVirtex; East-West Interconnections

I. INTRODUCTION

The basic Internet technologies developed over decades are becoming more difficult to meet increasingly diverse requirements. In general, only vendors can apply new functions and protocols into legacy network equipment. Therefore, it is hard to implement virtualized network functions for cutting-edge technologies such as security, cloud computing, and big data.

Recently, Software Defined Network (SDN) has been proposed as a clean-slate approach to provide network programmability solutions. The main difference between SDN and legacy networks is that the former splits the control plane and the data plane in legacy network equipment. The control plane can be deployed to a stand-alone controller for developing customized network services and applications. Taking advantage of programmable open architecture, SDN can satisfy various customized requirements, enhance and create innovative networking services and applications, increase utilization of bandwidth, enhance network engineering, enable optimization, and reduce production costs [1], etc.

As more and more SDN domains are deployed, it is very difficult to ignore the interconnections between legacy and SDN networks. In other words, it is extremely important to exchange information between these two different types of networks. However, the specification of an east-west interface is not defined by Open Networking Foundation (ONF) [2] and it is still an open issue at this time.

In order to resolve this problem, we propose to utilize NSI (Network Service Interface) [3] as the east-west interface. NSI is a standard protocol defined by Open Grid Forum (OGF) that manages and allocates network resources among legacy network domains. Users and edge processes can request connections using the agent software NSA (Network Service Agent).

In our previous research, we used NSI to implement an east-west interface among SDN domains [4], and integrated it with our SDN inter-domain topology and flow viewer [5][6][7]. However, this approach does not focus on the communications among NSI-managed legacy networks and SDN networks. Therefore, in this paper, we extend NSI east-west interface and integrate it with OpenVirtex [8] to enable the communications between legacy networks and SDNs. We also plan to connect with our collaborative organization, the International Center for Advanced Internet Research (iCAIR) [9] and implement the system over AutoGOLE testbed [10] to demonstrate this interconnection.

The remainder of this paper is organized as follows. Section 2 introduces the research background, including OpenNSA and OpenVirtex. The implementation of information transformation between NSI and OpenVirtex is described in Section 3. Experiments and results are demonstrated in Section 4. The conclusions are given in the last section.

II. BACKGROUND KNOWLEDGE

As the technologies of cloud computing develop and proliferate, the demands for network virtualization are emerging. Because of the programmability of SDN, the dynamic provisioning of virtualized networks can be easily achieved. In this paper, OpenVirtex is deployed to provide SDN-level virtualization within SDN domains. In order to interconnect among heterogeneous network domains, we employ NSI for east-west communications. We briefly introduce the related technologies as follows.

A. OpenVirtex

OpenVirtex (OVX) [11] is a tool for constructing virtualized networks in SDN environments. It allows multiple tenants to use the same network infrastructure.

Through address, topology, and control virtualization by OpenVirtex, all tenants can define and manage their own virtualized network topology. Administration of the virtualized networks can be customized using this approach and the flows of all tenants can be separated.

OpenVirtex is installed between physical network equipment and network controllers. Using OpenVirtex, each tenant has its own controller with a secure channel through the OpenFlow protocol. For each tenant, OpenVirtex can be viewed as network equipment. In contrast, for physical network equipment, OpenVirtex, which is connected via a channel through OpenFlow protocol, can be viewed as a controller. Figure 1 illustrates the architecture of OpenVirtex. The Network Embedder module in OpenVirtex provides the API of JSON RPC for receiving requests for constructing virtualized networks from tenants. Also, it maintains a mapping table from physical to virtualized networks.

Address virtualization is the key component of OpenVirtex. In order to allow all tenants to use IP address flexibly, tenants are mapped to virtualized IP addresses within OpenVirtex backbone. The virtualized network topology can be gathered from LLDP packets of tenants. Since modifying IP addresses is not generally supported by hardware switches, we modified OpenVirtex to separate tenants with vlan IDs [11]. Hence, the modified OpenVirtex can be deployed over the physical backbone network.

B. NSI and OpenNSA

NSI is a protocol standard for the network service layer proposed by Open Grid Forum (OGF) [12]. It is applied for resource sharing, topology interchanging, and dynamic network service allocation among network domains [13]. In general, users deploy an agent called Network Service Agent (NSA) to request inter-domain connections. NSA can take three different roles according to different operation modes described as follows.

- Ultimate Request Agent (uRA): requesting network service
- Aggregator (AG): intermediate that providing network services across multiple domains
- Ultimate Provider Agent (uPA): providing the network service

Taking the scenario in Figure 2 as an example, when users need to construct dedicated links and bandwidth, a request is issued by uRA. Next, AG starts to discovert the uPAs that satisfy the requirements. If a local uPA cannot satisfy the requests, the requests will be forwarded to neighbor network domain. This process does not stop until a uPA that can satisfy the requirements is found.

There are various implementations of NSA, such as OpenNSA[14], OSCARS [15], G-Lambda [16], AutoBHAN [17], etc. In this paper, we choose OpenNSA, an open-source implementation of NSI agent developed by NORDUnet. We

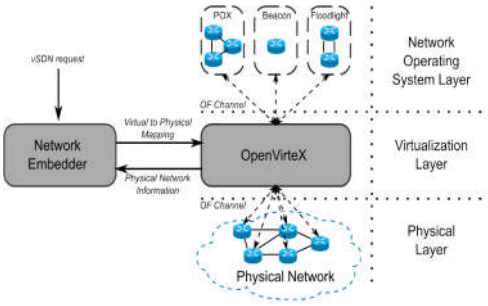


Fig. 1. OpenVirtex system architecture [18]

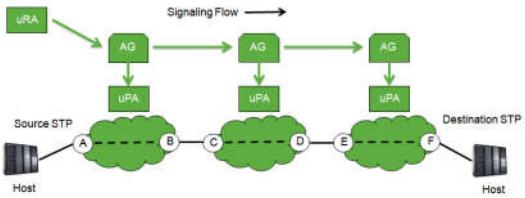


Fig. 2. Scenario of NSA use case [19]

extend OpenNSA to handle the topology exchange and connection request for OpenVirtex-based SDN networks.

III. DESIGN AND IMPELMENTATION

In this section, we propose a solution to information exchanges between SDN and legacy networks. With our implementation, based on OpenVirtex and OpenNSA, dynamic inter-domain connections can be provisioned. We briefly introduce our design concepts first and then describe implementation details.

A. Design Concepts

In our previous work, we have implemented a system which consists of a modified NSI and SDN controller to enable the message exchanging function between SDN controllers. However, the connection service is not supported for SDN domains. In this paper, we employ OpenVirtex as SDN controller to provide virtual networks and we develop two new services for OpenNSA as follows.

- Topology information exchange service, and
- Connection request service

The two services are demonstrated in Figure 3(A) and (B), respectively. There are four SDN domains A, B, C and D, that controlled by Ctrl A, Ctrl B, Ctrl C, and Ctrl D, respectively. We also deploy NSA A, NSA B, NSA C, and NSA D servers over these domains as NSI agents. Each NSA server exchanges messages with its own controller via North Bound Interface (NBI). There is another legacy domain E controlled by NSA E. We assume that each NSA maintains a peer list that contains the information of its neighbor NSAs.

In Figure 3(A), the process of topology exchange service of NSA A is listed as follows.

Step 1. NSA A asks its controller, Ctrl A, to get the topology information of domain A. The other NSA servers also do the same thing. Next, NSA A stores this topology information and obtains all its peers (B and E) from peer list and put them into a temporary visit list.

Step 2. According to the visit list, NSA A traverses all the peers, NSA B and NSA E, to request topology information of domains, which are controlled by them. It also obtains peer lists of NSA B (A, C and D) and E (A) and then put them into the visit list.

Step 3. After NSA A gets all data from NSA B and NSA E, the topology information will be stored. It will examine the new entries in the visit list (C and D) and traverse them in the next step.

Step 4. NSA A visits NSA C and NSA D to get topology information and put their peer list into the visit list. Since there are no unvisited entries, the process of topology exchange is ended.

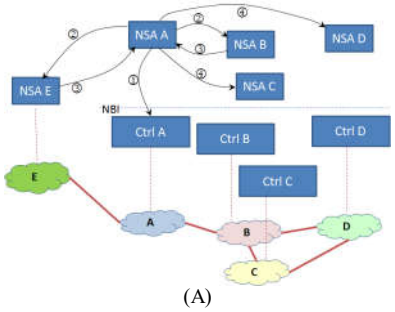
Assume there is a connection request from client X in domain E to client Y in domain D. The process of connection request service is listed as follows in Figure 3(B).

Step 1. NSA E receives the request and compute the path X→E→A→B→D.

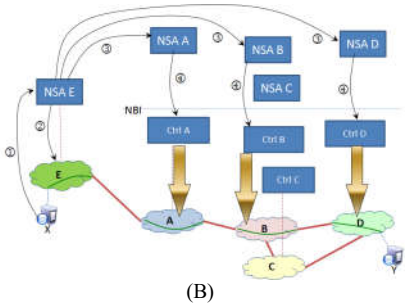
Step 2. NSA E creates and configures a routing path within domain E for X.

Step 3. NSA E sends this connection request to NSA A, B, and D concurrently.

Step 4. NSA A, B and D ask Ctrl A, B, and D respectively to create a routing path within their own domains. Finally, X is able to send flows to Y following this path.



(A)



(B)

Fig. 3. (A) Topology Information Exchange Service (B) Connection Request Service

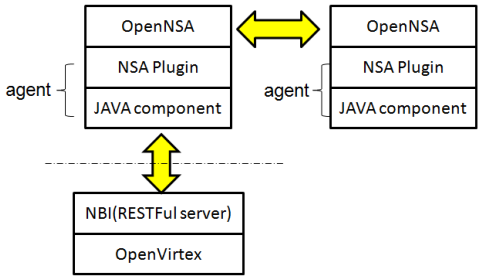


Fig. 4. System Architecture on OpenNSA

B. Implementation of topology information exchange

According to our design concepts, we setup the system consisting of OpenNSA and OpenVirtex. For the implementation of the topology information exchange, there are three key problems to be solved: implementation of NBI, the communication mechanism between OpenNSA sites, and the API for OpenNSA topology. Since OpenVirtex does not provide NBI functions, we implement a RESTFul Server [20] to act as the NBI. For the communication mechanism between OpenNSA sites, we utilize the original NSI handshake mechanism. As to API for the OpenNSA topology, we implement an agent on OpenNSA. This agent can not only exchange messages among OpenNSA sites, but also exchange data with other user applications.

Figure 4 depicts the system architecture using OpenNSA. The agent for OpenNSA consists of a NSA Plugin and JAVA component. Since the JAVA component cannot communicate with OpenNSA directly, we have to implement an NSA plugin as a middleware component. A JAVA component can obtain the peer list from the NSA plugin and put them into a temporary visit list. Also, the JAVA component can invoke the NSA plugin to gather topology information and request peer lists from other OpenNSA peers.

The JAVA component has four functions, OpenVirtex polling, OpenNSA polling, data store, data receiving and transmission. OpenVirtex polling communicates with a RESTFul server of OpenVirtex to obtain SDN topology periodically; OpenNSA polling notifies NSA Plugin to obtain the topology of peers via OpenNSA; the data store function stores the visit list and topology information in a customized data structure. Finally, the data receiving and transmission function performs the data exchange with OpenVirtex, OpenNSA or other user applications.

Figure 5 is the flowchart of the topology information exchange. When OpenNSA is initialized, the peer list will be sent to JAVA component via NSA Plugin. After that, JAVA component will communicate with OpenVirtex and NSA Plugin periodically. For the communication with OpenVirtex, the JAVA component sends topology information requests to NBI of OpenVirtex. OpenVirtex then fetches SDN topology information and replies via NBI. For the communication with the NSA Plugin, the JAVA component asks the NSA plugin to traverse all peers according to visit list to obtain their topology and peer lists.

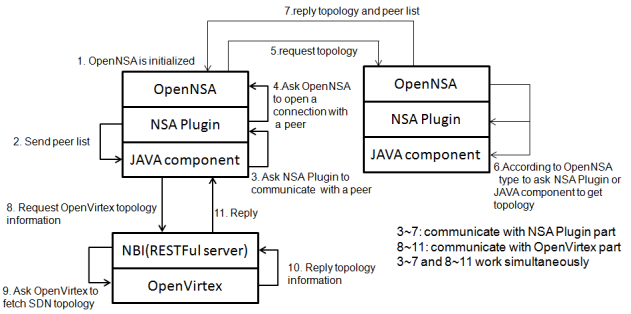


Fig. 5. Flowchart of topology exchange

When there is a request for topology and peer list from peer OpenNSAs, the JAVA component will obtain the local topology information. Such information can be gathered from NSA Plugin if the domain consists of legacy layer 2 switches. Otherwise, for SDN domains, topology information resides in JAVA component. After that, topology information and peer list are sent to the requesting OpenNSA as a reply. Finally, the requesting OpenNSA passes this information to the JAVA component to handle and store. According to the above processes, each OpenNSA will be able to calculate the global topology.

C. *Implementation of Connection request function*

OpenNSAs can request connections with each other for legacy Layer 2 networks. However, there is no communication mechanism for OpenNSA to obtain information from OpenVirtex-based SDN networks. Hence, we design a translator in OpenVirtex NBI to map commands between OpenVirtex and OpenNSA.

Figure 6 shows the original OpenNSA connection request process as follows.

Step 1. When OpenNSA receives a connection request, it first analyzes this request and then starts the connection service.

Step 2. The connection service will invoke a specific backend

Step 3. The specific backend sends the configuration command to Layer 2 switches through the SSH protocol.

Step 4~6. The configuration results will be sent back to requester.

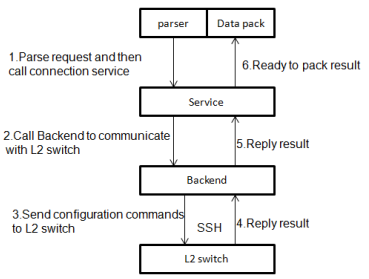


Fig. 6. Flowchart of original OpenNSA connection request process

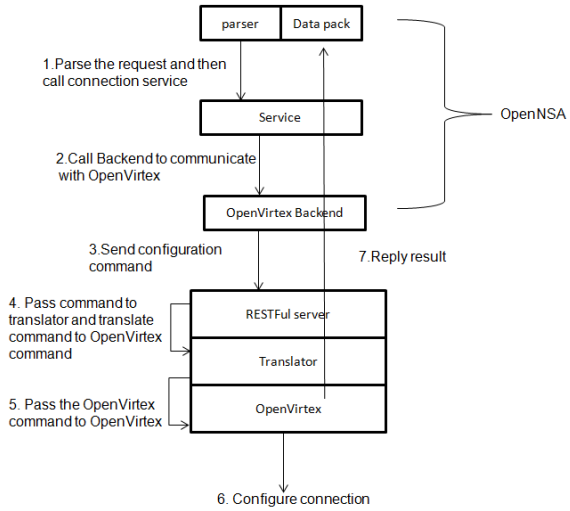


Fig. 7. Flowchart of the enhanced connection request process

Since the original OpenNSA backend does not have a communication mechanism with OpenVirtex, we implement a new backend to communicate with OpenVirtex through NBI. Figure 7 shows the processes of our implementation. All steps are explained as follows.

Step 1. When OpenNSA receives a connection request, it first analyzes this request and then starts the connection service.

Step 2. The connection service will invoke the OpenVirtex backend.

Step 3. The OpenVirtex backend sends the configuration command to OpenNSA NBI.

Step 4. NBI gets the configuration command, and then passes to translator. The translator parses the commands and then translates them to OpenVirtex commands.

Step 5. The translator passes the OpenVirtex configuration commands to OpenVirtex.

Step 6. OpenVirtex executes the configuration commands.

Step 7. The configuration results will be sent back to requester.

IV. EXPERIMENTS

In this section, we use two different environments to test the function of connection setup and topology information exchange. The testing infrastructure for connection setup is shown as Figure 8. In control plane, OpenVirtex controls two SDN switches, SDN A and SDN B. Two hosts A and B, with static IP addresses of 10.0.0.1 and 10.0.0.2, are connected to the 2nd port of SDN A and SDN B, respectively.

The testing process was designed to make sure that 10.0.0.1 can ping 10.0.0.2 through two SDN switches successfully. The whole testing processes are listed as follows and shown in Figure 9.

Step 1. OpenNSA sends a connection request to OpenVirtex.

Step 2. When OpenVirtex receives the connection request from OpenNSA, it will assign flows between SDN A and SDN B. A VLAN ID (100) is generated for this dedicated connection.

Step 3. Host A pings Host B successfully as shown in Figure 10.

Next, we test the function of topology information exchange. The testing infrastructure is shown in Figure 11. There are 3 network domains twaren1.nchc.org (denoted as TWAREN1), twaren2.nchc.org (denoted as TWAREN2), and twaren3.nchc.org (denoted as TWAREN3). TWAREN2 connects to TWAREN1 and TWAREN3 in separated links. TWAREN1 and TWAREN2 are SDN domains while TWAREN3 is a legacy Layer 2 domain.

After the process of topology information exchange finished, we obtain the topology information from OpenNSA1, shown in Figure 12(a) and (b). Figure 12(a) depicts the abstract information of all domains. There are 3 network domains, links between domains, and types of all network domains. The detailed information within each domain can be found in Figure 12(b).

We also developed a Web-based GUI viewer, which can transform text topology information into a graphical view. Figure 13 displays the names and types of all network domains, including OpenVirtex or Layer 2 switch domains.

In Figure 13, when users click on the OpenVirtex node, the detailed topology graph of the selected node will be displayed. As shown in Figure 14, there are 3 switches and their links. Each switch is labeled with their data path ID (DPID).

In addition, when users click the Layer 2 switch node, the detailed information of the selected Layer 2 switch will be displayed, as shown in Figure 15. When users click on a number, which is the port id, then its detailed information is shown in the right-hand side for network administrators to manage.

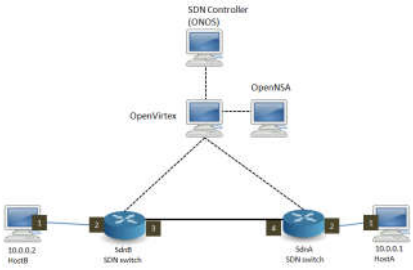


Fig. 1. Backend communicate with OpenVirtex

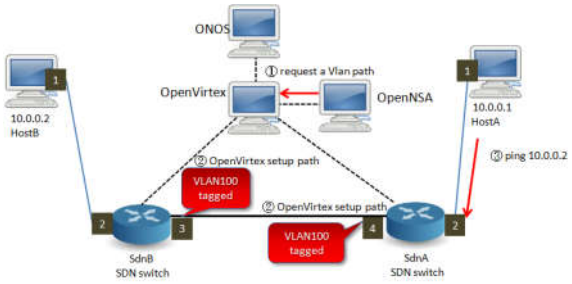


Fig. 2. Flows of testing process

```
nchclab@HC-NODE:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1059 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=555 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=5.90 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=5.91 ms
```

Fig. 3. Testing results

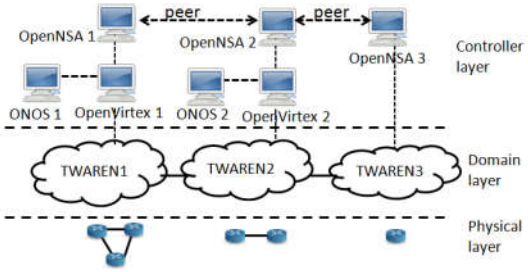


Fig. 4. Infrastructure of topology information exchange


```
{"node":[{"nodetype":"openvirtex","name":"twaren1.nchc.org"},
         {"nodetype":"openvirtex", "name":"twaren2.nchc.org"},
         {"nodetype":" L2switch", "name":"twaren3.nchc.org"}],
 "link":[{"dst":"twaren2.nchc.org","src":"twaren1.nchc.org"},
         {"dst":"twaren1.nchc.org","src":"twaren2.nchc.org"},
         {"dst":"twaren3.nchc.org","src":"twaren2.nchc.org"},
         {"dst":"twaren2.nchc.org","src":"twaren3.nchc.org"}],
 "type":"abstract"}
```


(a) Abstract topology data of all domains

twaren1.nchc.org

```
{"json":[
{"switches":[{"dpid":"cc:4e:24:d1:12:80:00:00"},{"dpid":"cc:4e:24:d1:19:00:00:00"},{"dpid":"cc:4e:24:d1:0c:00:00:00"}],
 "sitename":"twaren1.nchc.org",
"links":[{"linkId":277,"dst":{"port":"5","dpid":"cc:4e:24:d1:0c:00:00:00"},"src":{"port":"4","dpid":"cc:4e:24:d1:19:00:00:00"}},
         {"linkId":281,"dst":{"port":"5","dpid":"cc:4e:24:d1:19:00:00:00"},"src":{"port":"5","dpid":"cc:4e:24:d1:12:80:00:00"}},
         {"linkId":278,"dst":{"port":"4","dpid":"cc:4e:24:d1:19:00:00:00"},"src":{"port":"5","dpid":"cc:4e:24:d1:0c:00:00:00"}},
         {"linkId":279,"dst":{"port":"4","dpid":"cc:4e:24:d1:0c:00:00:00"},"src":{"port":"3","dpid":"cc:4e:24:d1:12:80:00:00"}},
         {"linkId":282,"dst":{"port":"5","dpid":"cc:4e:24:d1:12:80:00:00"},"src":{"port":"5","dpid":"cc:4e:24:d1:19:00:00:00"}},
         {"linkId":280,"dst":{"port":"3","dpid":"cc:4e:24:d1:12:80:00:00"},"src":{"port":"4","dpid":"cc:4e:24:d1:0c:00:00:00"}}]}],
```

twaren2.nchc.org

```
{"switches":[{"dpid":"00:00:00:00:00:00:00:02"},{"dpid":"00:00:00:00:00:00:00:01"}],
 "sitename":"twaren2.nchc.org","openvirtex",
 "links":[{"linkId":37,"dst":{"port":"2","dpid":"00:00:00:00:00:00:00:02"},"src":{"port":"2","dpid":"00:00:00:00:00:00:00:01"}},
          {"linkId":36,"dst":{"port":"2","dpid":"00:00:00:00:00:00:00:01"},"src":{"port":"2","dpid":"00:00:00:00:00:00:00:02"}}]},
"type":"detail"}
```

twaren3.nchc.org

```
{"interfaces":[{"remote_out":"-", "port_type":"ethernet", "bandwidth":1000, "remote_port":"-","name":"ofport1",
               "label":[{"min":1779,"max":2799,"labelType":"vlan"}],"interface":"ge-1/1/1","remote_in":"-","authz":"[-]","remote_network":"-"},

              {"remote_out":"-","port_type":"ethernet","bandwidth": 1000,"remote_port":"-","name":"ofport2",
               "label":[{"min":1779,"max":2799,"labelType":"vlan"}],"interface":"ge-1/1/2","remote_in":"-","authz":"[-]","remote_network":"-"}],
"sitename":"twaren3.nchc.org","openvsaType":"L2switch"},
```


(b) Physical topology data

Fig. 5. Experimental results

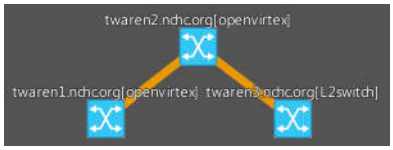


Fig. 6. Snapshot of overall abstract infrastructure

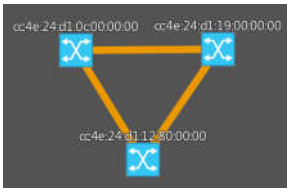


Fig. 7. Snapshot of detail information of the OpenVirtex node

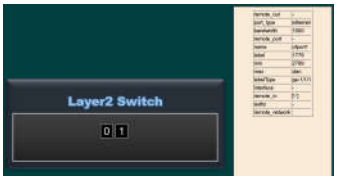


Fig. 8. Detailed information of a Layer 2 switch node

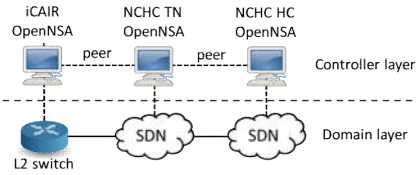


Fig. 9. Architecture integrated with iCAIR

V. CONCLUSIONS

In this paper, we describe a modified OpenNSA and the implementation of an agent to enable OpenNSA to gather the topologies of SDN and legacy networks. We can dynamically allocate virtual networks across SDN and legacy network domains. Users can also view both global and local topologies with a Web-based GUI viewer.

In the future, we will extend the topology to iCAIR in order to connect with AutoGOLE, which is a NSI testbed with more than 20 international organizations. As shown in Figure 16, iCAIR works as an aggregator of AutoGOLE and we will setup a peer between NCHC and iCAIR for NSI exchanges. This will enhance the connections between SDN and legacy networks, and will promote cooperation among international organizations.

REFERENCES

[1] S. Ortiz, "Software-defined Networking: On the Verge of a Breakthrough?" Computer, vol. 46, no. 7, pp. 10-12, Jul. 2013.

[2] Open Networking Fundation, https://www.opennetworking.org/

[3] Network Service Interface (NSI), https://redmine.ogf.org/projects/nsi-wg

[4] W. Y. Huang, H. L. Lee, and T. L. Liu, "Application and Implementation of NSI for Information Transportation on SDN Networks," Proceedings of TANET 2015 – Taiwan Academic Network Conference, Nantou, Taiwan R.O.C., Oct. 2015.

[5] W. Y. Huang, J. W. Hu, S. C. Lin, T. L Liu, P. W Tsai, C. S. Yang, F. I. Yeh, J. H. Chen, and J. Mambretti, "Design and Implementation of an Automatic Network Topology Discovery System for the Future Internet across Different Domains," Proceedings of the 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 903-908, Mar. 2012.

[6] W. Y. Huang, J. W. Hu, T. Y. Chou, and T. L. Liu, "Design and Implementation of Real-Time Flow Viewer across Different Domains," Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA) , pp. 619-624, Mar. 2013.

[7] W. Y. Huang, T. Y. Chou, J. W. Hu, T. L. Liu, "RESTFul API and HTML5-based interdomain end-to end topology and flow viewing system in SDN networks" Proceedings of TANET 2014 – Taiwan Academic Network Conference, Kaohsiung, Taiwan R.O.C., Oct. 2014.

[8] A. Al-Shabibi, M. D. Leenheer, M. Gerola, A. Koshibe, E. Salvadori, G. Parulkar, and B. Snow. OpenVirteX: Make your virtual sdns programmable. In Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software De ned Networking (HotSDN 2014), pp. 25-30, 2014.

[9] International Center for Advanced Internet Research, http://www.icair.org/

[10] AutoGOLE, http://dashboard.lab.uvalight.net/overview

[11] A. Al-Shabibi, M. D. Leenheer, M. Gerola, A. Koshibe, E. Salvadori, G. Parulkar, and B. Snow. OpenVirteX: Make your virtual sdns programmable. In Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software De ned Networking (HotSDN 2014), pp. 25-30, 2014.

[12] Open Grid Forum (OGF), http://www.ogf.org/

[13] H. L. Lee, Jim Hao Chen, Fei I. Yeh, M. Chen, T. L. Liu, "Layer2 SDX: SDN exchanging center," Proceedings of TANET 2015 – Taiwan Academic Network Conference, Nantou, Taiwan R.O.C., Oct. 2015.

[14] OpenNSA, https://github.com/NORDUnet/opennsa/

[15] OSCARS, https://www.es.net/engineering-services/oscars/

[16] A. Takefusa, et. al, "G-lambda: coordination of a grid scheduler and lambda path service over GMPLS Source," Future Generation Computer Systems ,Vol.22 , Issue 8, pp. 868 – 875, October 2006.

[17] M. Büchli, at al., "Deliverable DJ.3.3.1:GÉANT2 Bandwidth on Demand Framework and General Architecture", GÉANT, 2005.

[18] J. MacAuley et. al, "Network Service Interface Signaling and Path Finding", GWD-I NSI-WG, December 2014.

[19] J. W. Hu, H. M. Tseng, and T. L. Liu, "Implementation of Dynamic Virtualized Network Provisioning using OpenFlow," Proceedings of TANET 2015 – Taiwan Academic Network Conference, Nantou, Taiwan R.O.C., Oct. 2015.

[20] REST API, http://www.restapitutorial.com