Proceedings of the APAN – Research Workshop 2017
ISBN 978-4-9905448-7-4

Design and Development of the Reactive BGP peering in Software-Defined Routing Exchanges

Hao-Ping Liu, Pang-Wei Tsai, Wu-Hsien Chang and Chu-Sing Yang

***Abstract*—The Software-Defined Networking (SDN) is considered to be an improved solution for applying flexible control and operation recently in the network. Its characteristics include centralized management, global view, as well as fast adjustment and adaptation. Many experimental and research networks have already migrated to the SDN-enabled architecture. As the global network continues to grow in a fast pace, how to use SDN to improve the networking fields becomes a popular topic in research. One of the interesting topics is to enable routing exchanges among the SDN-enabled network and production networks. However, considering that many production networks are still operated on legacy architecture, the enabled SDN routing functionalities have to support hybrid mode in operation. In this paper, we propose a routing exchange mechanism by enabling reactive BGP peering actions among the SDN and legacy network components. The results of experiments show that our SDN controller is able to mask as an Autonomous System (AS) to exchange routing information with other BGP routers.**

***Index Terms*—Software-Defined Networking, OpenFlow, BGP, Software-Defined Routing.**

I. Introduction

As the evolution of the network, there are more and more requirements for new protocol testing or devices update in all network environments. However, under current network architecture, it takes both huge time and financial cost to carry out these tasks. For example, routers play an indispensable role in the environment such as data centers or backbone networks in which even shutdown a little while for update will result in an unpredictable loss. Besides, network management and performance tuning is quite challenging because that network devices are usually vertically-integrated black boxes [1]. The development of devices is mastered by the vendors, whereas customers can only passively wait for the expensive and inflexible products provided by them.

The above-mentioned example shows the limit of the legacy network. Eventually, networks with this closed architecture become ossified [2] and lead to a bottleneck for the progress of the real world. Yet the emergence of Software-Defined Networking (SDN) [3] provides a solution to this problem. SDN brought the concept that separating the data plane and the control plane of a network. Allowing network operators to directly operate networks in a centralized manner with an independent controller in the control plane. In addition, the devices in the data plane such as switches, just simply perform the forwarding of packets according to the policies set by the SDN controller. There are already some ongoing researches and implements of SDN [4], and Figure 1 shows the most popular referred SDN architecture [5].

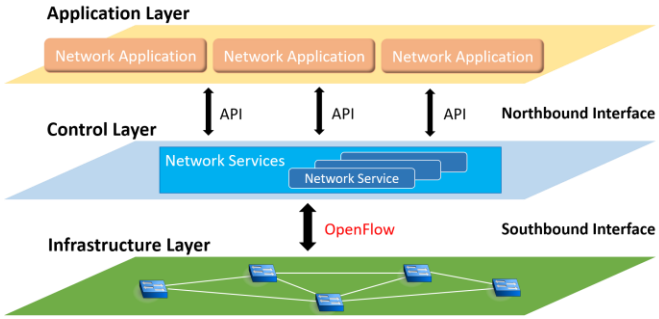

Fig. 1. The logical view of a SDN architecture [5].

In this SDN architecture, developers can easily deploy their innovations just by programming applications in the application layer. The core network services in the control layer interact with the applications through the Northbound Interface such as a RESTful Application Programming Interface (API) [6], and dynamically modify the forwarding behavior of the network devices in the infrastructure layer through the Southbound Interface, that is, the OpenFlow protocol [2]. A device in the infrastructure layer maintains flow tables which are composed of several flow rules. A flow rule contains a match field and an instruction field. The match field defines a series of characteristics of a packet, and the instruction field defines several actions to manipulate a matched packet. When a packet comes into a data plane device, a pipeline procedure starts to compare the incoming packet through the match field of these flow rules, and finally figure out the output port or other operations to this packet.

Hao-Ping Liu, Pang-Wei Tsai, Wu-Hsien Chang and Chu-Sing Yang are with the Institute of Computer and Communication Engineering, Department of Electrical Engineering, National Cheng Kung University, Taiwan (email: alen6516@gmail.com, pwtsai@ee.ncku.edu.tw, shane50306@gmail.com, csyang@ee.ncku.edu.tw)

Through centralizing the control intelligence and modifying the flow tables, SDN breaks the monopoly of the vender-dependent network appliances by using commodity hardware with a free, open source Network Operating System (NOS) [7]. Network hardware and software can then evolve independently, and the function developers turn to just focus on the exploitation of their new ideas without concerning about the difficulty in the subsequent deployment. SDN augments the programmability and virtualization while simultaneously simplifies the configuration and troubleshooting of networks. Though many challenges still in processing, SDN has been considered as the revolution to the current networking. Besides the newly deployment of SDN in wide-area networks [8], the conversion from legacy IP networks to the SDN or hybrid networks is also an ongoing research issue now [4]. The challenge is, as Sezer et al. [9] has pointed out, it requires a hybrid infrastructure in which the legacy and SDN-enabled network nodes can operate in harmony. Such interoperability needs SDN communication interfaces to provide backward compatibility with the existing IP routing to retain the connection between the SDN network and other legacy IP networks. To solve this challenge, Lin et al. [10], Rothenberg et al. [12], and Thai et al. [13] have mentioned the utilization of BGP [14]. Due to its stable and widely deployed in current IP networks, keeping using BGP during the gradual update is more practical.

In this paper, we design a virtual BGP entity that combines a reactive BGP peering mechanism to the SDN control logic. With this design, the SDN domain is able to act as a transit AS which can reactively build BGP sessions with external legacy networks and propagate the routing information as well as the inter-domain IP flows from one external network to the others. The remainder of this paper is organized as follows. Section II gives a brief introduction to the related works. Section III demonstrates the comprehensive design of our system. Section IV brings an experiment to verify the functionality of our implementation. Section V gives the discussion over the experimental results and indicates the potential improvements. Finally, a conclusion of this paper is provided in Section VI.

II. Related Work

There are already some researches and implementations about designing a BGP-enabled SDN framework or a hybrid system that associating SDN with IP routing. These works bring about many great ideas, and this section gives a brief introduction to them.

2.1 RouteFlow [15] uses virtual machines (VMs) to control the behavior of OpenFlow switches by mapping each active ports of switches to a virtual network interface on VMs one by one. These VMs run open source routing protocols such as BGP and Open Shortest Path First (OSPF) [16], and form a virtual topology by connecting with each other. Therefore, VMs can exchange the routing information and control the behavior of the switches as if they are running a distributed control plane.

2.2 Open Source Hybrid IP/SDN networking (OSHI) [17] combines the regular IP routing with SDN-based forwarding and provides a hybrid IP/SDN network node on Linux. This hybrid node uses Quagga software [18] for OSPF routing and Open vSwitch software [19] for OpenFlow-based switching. Packets can be routed in regular IP method or SDN-based paths (SBPs) alternatively by considering the headers at different protocol levels. Evaluations are also presented to display the performance of SBPs.

2.3 Hong et al. [20] propose a hybrid system consisting of both legacy forwarding devices and programmable SDN switches. They study how to satisfy a variety of traffic engineering goals such as load balancing or fast failure recovery during the incremental deployment of SDN. An evaluation on real ISP and enterprise topology is also presented with discussion.

2.4 SDN-IP [10] and BTSDN [11] both propose a peering manner between SDN and IP networks. In their SDN context, several legacy BGP routers are attached to the OpenFlow switches. These BGP routers are responsible for peering with the external IP networks. The routing information received by these routers in the data plane should be synchronized to the SDN-IP application in the SDN controller via an out-of-band control link as Figure 2 shows. This approach utilizes legacy BGP routers as a BGP proxy for the SDN domain. However, considering the spirit of SDN, that is, centralizing all configuration and control of the network, we think removing the proxy BGP routers and just integrating the BGP control mechanism into the SDN/OpenFlow architecture is more intuitive. This idea then turns out to be our motive.

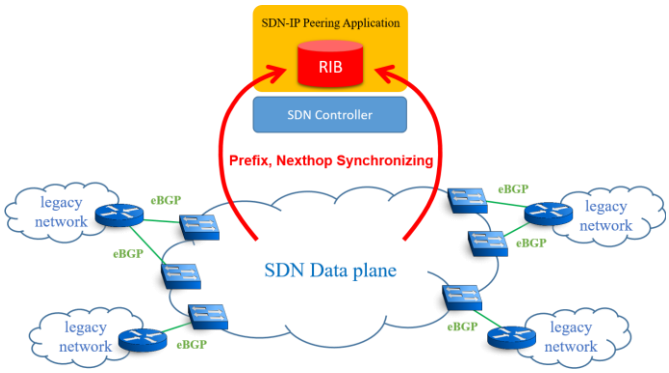

Fig. 2. The architecture of SDN-IP network peering [9].

III. System Design

Since the biggest difference between SDN-IP and our system is that we combine the BGP capacity to the SDN control logic rather than using a legacy BGP router in the data plane as a proxy, the BGP messages from neighbors are actually encapsulated as OpenFlow packet-in messages and sent to the controller by switches. Similarly, the replies from the controller are also encapsulated as OpenFlow packet-out messages and sent to the corresponding switch which will forward it to the corresponding neighbor afterward. The details of the operation will be described in the following article. In this chapter, part A gives an overall view of the scenario. Part

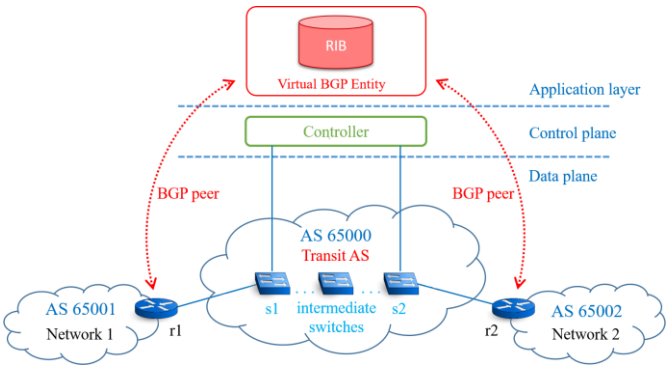

Fig. 3. The scenario of our approach.

B describes how to achieve the peering mechanism by the cooperation of modules designed by us. Part C shows the receipt, handling and advertisement of the routing information as well as the subsequent update of Routing Information Base (RIB). Finally, part D describes how we fulfill the requirement of software-defined routing for IP traffics over the SDN network.

A. *Overview*

Our approach simplifies the peering mechanism from SDN-IP by removing the legacy BGP routers in the SDN data plane. Figure 3 describes the scenario that two legacy networks with AS number 65001 and 65002 connect to a SDN network with AS number 65000. Each external network has an edge BGP router (named r1 or r2) which are used to peer with the SDN domain. In the SDN domain, s1 and s2 are OpenFlow-enabled switches that connect to r1 and r2 respectively, and the remaining OpenFlow-enabled switches in the SDN domain are named as intermediate switches. All of these switches are controlled by a SDN controller.

In the controller, we leverage virtual network interfaces and several programming modules to constitute a virtual BGP entity to handle the procedure of the External BGP (eBGP) sessions. Figure 4 shows all of the modules used by the virtual BGP entity with their organization. After an initialization by Main module, every BGP control message from the neighbors (i.e., r1 and r2) will match a proactively installed table-miss flow rule in the data plane and then be encapsulated as an OpenFlow packet-in message to the controller. Protocol Handler module is responsible for parsing the BGP packets in these packet-in messages and deciding the next step, such as replying a BGP open message or a BGP keep-alive message to start or maintain an eBGP session. In this manner, our virtual BGP entity can properly interoperate with the neighbors.

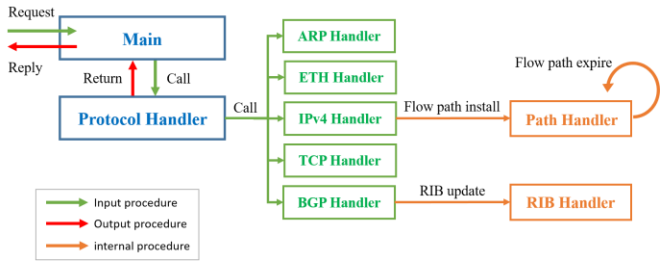

Fig. 4. The organization of our modules.

B. *Peering Mechanism*

To achieve the BGP peering, what we need to handle is the entire control of the communication. So our Protocol Handler module must be able to respond correctly for different kinds of requests including ARP, TCP handshake and BGP queries. In the initialization, Main module acquires neighbors' information by reading a configuration file set in advance. Then the system gets ready to parse the incoming packets and starts waiting for the requests from the external BGP routers. To respond to the layered design of TCP/IP suite, our Protocol Handler is also designed in a layered manner. For an incoming packet from a neighbor, Protocol Handler judges and calls submodules, including ARP Handler, ETH Handler, IPv4 Handler, TCP Handler and BGP Handler, to handle packet headers at different protocol level, and generates the appropriate reply. Afterward, Main module assigns the corresponding switch to send out this reply back to the neighbor. This is how a control packet from neighbors be handled.

C. *RIB Update*

We need to update the RIB of the virtual BGP entity once a BGP update message is recognized by Protocol Handler. An RIB update event will be triggered and inform BGP Handler to take out the information, including Network Layer Reachability Information (NLRI), path attributes and withdrawn routes (if any) from the packet, then RIB Handler uses this information to insert or delete prefixes in the local RIB. Finally, after the RIB update, our BGP entity should also advertise this update information to the other neighbors to continue the information propagation.

D. *Software-defined Routing Mechanism*

Our virtual BGP entity has learned and propagated the routing information among neighbors after RIB updates. Each external BGP router regards our virtual BGP entity as the next hop to the others. For the external IP flows from one external BGP router to another, that is, the inter-domain IP flows, we design a software-defined routing mechanism to arranges a path inside the SDN data plane. This path, called flow path, is composed of a series of switches that can forward the IP traffics one switch by one switch over the SDN domain. However, OpenFlow switches can do nothing before flow rules have been installed to them. Also, the destination MAC address of the IP flows is still the MAC address of our virtual BGP entity. Some of above functionalities are based on our previous works [21][22]. The following descriptions will introduce how Path Handler module dynamically install and remove flow rules to achieve the routing of the external IP flows.

1) *Flow Path Installation*

To prepare a path between two corresponding neighbors for a new inter-domain IP flow. In current prototype we select the shortest path among the switches. The first packet of this IP flow causes a packet-in event to the controller and triggers Path Handler to install a series of flow rules to the switches along the selected flow path. Switches with these flow rules match the input port as well

as the destination IP prefix of the packets, and send out the matched packets to an output port. Traffics of the IP flows can be routed along this path by continuously matching these rules from one switch to the next switch and finally reach the corresponding neighbor.

2) *Layer 2 and Layer 3 Routing Mechanism*

Even though we have arranged a path for the inter-domain traffics from one neighbor to another neighbor, the question is that the destination MAC address of these traffics are still the MAC address of the virtual interface because the sender regards our virtual BGP entity as the next hop. If switches just forward them, packets will be dropped due to the wrong destination MAC address. So in order to satisfy layer 2 connection, we add a destination MAC rewriting action to the flow rule. As for satisfying layer 3 routing, we also add a Time To Live (TTL) value decreasing action to this flow rule. Eventually, after a packet matches this flow rule, the packet's TTL value will be subtracted by 1 and the destination MAC address will be changed to the MAC address of another neighbor just as how a router routes a IP packet.

3) *Flow Path Elimination*

Besides installation, we still need a mechanism to eliminate the useless flow rules to avoid the excessive entries on the switches. OpenFlow provides an idle timeout control for the removal of a flow rule. We use this feature and set a proper timeout period for flows in different priority levels. Then the flow rule will be automatically eliminated from the flow table if no packets match it before the timer expires. Finally, combining the flow path installation with elimination mechanism, we can dynamically insert and remove flow paths in the SDN domain.

IV. Experiment Results

At current stage, we devoted to the architecture design and implement the first prototype. The following experiments are designed to verify the feasibility of our idea, including the handling of the BGP sessions as well as the software-defined routing mechanism for the inter-domain IP flows.

We adopt Mininet [23] as the network emulator for this experiment. Researchers can use Mininet to design a customized virtual network testbed on a single Linux kernel simply with its virtualization capability. We run the experimental topology shown in Figure 5 on Mininet. This topology shows three legacy IP networks with AS number 65001, 65002, and 65003 respectively. They all connect to a SDN domain with AS number 65000. There is a BGP routers in each legacy IP network (i.e., r1, r2, and r3) and each BGP router individually connects to a host in their domain (i.e., h1, h2, and h3). In the data plane of the SDN domain are three inter-connected OpenFlow switches (i.e., s1, s2, and s3) which link to a legacy network respectively. These OpenFlow switches are all controlled by single SDN controller in the SDN control plane.

For r1, r2, and r3, we adopt Quagga routing suite version 0.99.22.4 as the BGP software. Quagga is a network routing software suite that can provide a Unix-like system with

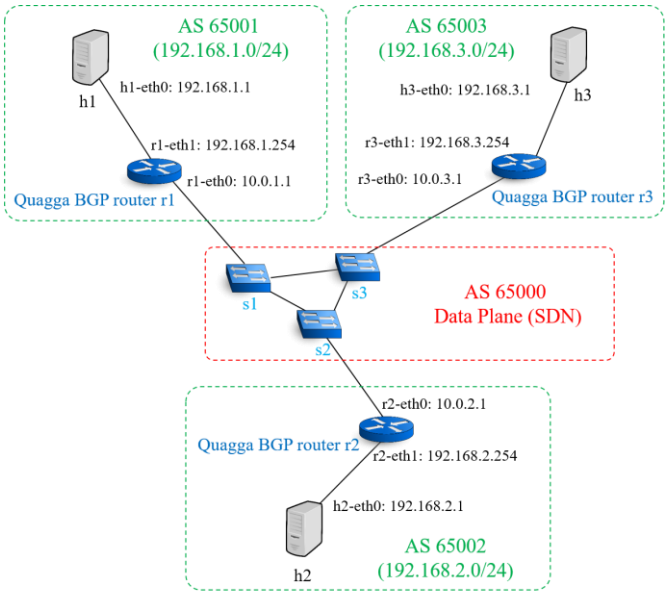

Fig. 5. Experimental topology.

```
mininet> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 r1-eth1
192.168.2.0     10.0.1.101      255.255.255.0   UG    0      0        0 r1-eth0
192.168.3.0     10.0.1.101      255.255.255.0   UG    0      0        0 r1-eth0
mininet> r2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 r2-eth0
192.168.1.0     10.0.2.101      255.255.255.0   UG    0      0        0 r2-eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 r2-eth1
192.168.3.0     10.0.2.101      255.255.255.0   UG    0      0        0 r2-eth0
mininet> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.3.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
192.168.1.0     10.0.3.101      255.255.255.0   UG    0      0        0 r3-eth0
192.168.2.0     10.0.3.101      255.255.255.0   UG    0      0        0 r3-eth0
192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 r3-eth1
```

Fig. 6. Routing tables of each external router.

```
mininet> h1 ping -c 3 h2
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=61 time=0.259 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=61 time=0.085 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=61 time=0.195 ms

--- 192.168.2.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.085/0.179/0.259/0.073 ms
mininet> h1 ping -c 3 h3
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=1 ttl=61 time=0.520 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=61 time=0.064 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=61 time=0.110 ms

--- 192.168.3.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.064/0.231/0.520/0.205 ms
mininet> h2 ping -c 3 h3
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=1 ttl=61 time=0.292 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=61 time=0.269 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=61 time=0.080 ms

--- 192.168.3.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.080/0.213/0.292/0.096 ms
```

Fig. 7. The successful Ping test between hosts.

multiple routing mechanisms. In the SDN domain, we adopt Open vSwitch version 2.0.2 implemented in Mininet as the switch software for s1, s2, and s3, and Ryu [24] version 4.10 as the SDN controller software. Ryu is a SDN framework based on Python module components. Ryu provides many well defined APIs that simplify the development of the

management and control in the SDN environment. Both Mininet and Ryu software are running on the same VM which is equipped with 2 processors, 4GB RAM, and runs Ubuntu-14.04-desktop-amd64 as the OS. This VM is managed by the VirtualBox software [25] running on a PC with 12GB RAM, intel core i7-4770 CPU, and Microsoft windows 10 as the OS.

To validate feasibility of our design, we start Ryu with our approach as an application to control this topology. After a little while for building BGP sessions and exchanging routing information, we check the routing table of the three external BGP routers. As the routing tables shown in Figure 6, every BGP router records the IP prefix of other ASes. Thus we confirm that our SDN domain can properly receive the BGP update messages from an external IP network and advertise these routes to the others. Then we do the Ping tests between the hosts in different legacy networks (i.e., h1, h2, and h3) to make sure the software-defined routing of IP traffics over the SDN domain. As Figure 7 that shows the successful Ping requests and replies, the software-defined routing mechanism of our approach is proved.

V. Discussion

The above experimental result illustrates that SDN domain with our approach is able to exchange routing information with neighbors. IP flows are also allowed to traverse the SDN domain with the software-defined routing mechanism. For the external BGP routers, the SDN domain performs just as same as a legacy BGP router. We have achieved the basic stitching between these two type of network paradigms. However, to become more practical, the BGP routing mechanism should still satisfy several requirements such as high capacity of RIB, fast IP lookup, high reliability and so on. We have not tested our system with the BGP routers in real internet environment. Scalability issues are predictable due to the restriction of the size of flow tables in the switches and the performance of single controller. Furthermore, there are still many topics worth studying by considering the advantages of SDN based on this approach. For example, now the flow paths are selected by only considering shortest path of the switches, we can design a best flow path selection algorithm by thinking of more conditions like switches' load or flow priority, and even design a flow migration mechanism to prevent a switch failure or link break. These are all potential issues which we have planned to implement in the future.

VI. Conclusion

In this paper, we design a suite of reactive BGP peering and a software-defined routing mechanism that can mask a SDN domain as a transit AS to propagate routing information and IP flows among the adjacent external networks. This design can increase the compatibility of SDN and legacy IP networks during the incremental deployment. To integrate BGP control into the SDN control logic, we design a virtual BGP entity in the SDN controller. By utilizing OpenFlow packet-in and packet-out messages, our system enables the SDN controller to exchange BGP messages with neighbors through the OpenFlow switches in the data plane. Besides, our approach also provides the software-defined routing mechanism for the inter-domain IP traffics. This mechanism arranges a flow path and enables the IP flows to traverse a SDN domain with the achievement of layer 3 IP routing by decreasing TTL value and layer 2 Ethernet delivery by rewriting the destination MAC address. In the end, the result of the experiment proves the feasibility of our approach as an application in the Ryu controller to be a transit AS that propagates the routing information and inter-domain IP traffics among multiple domains.

Acknowledgement

This research was supported in part by the Ministry of Science and Technology of Taiwan, under contracts No.104-2221-E-492-002-MY2 and 105-2218-E-001-001. Authors are grateful to the National Center for High-Performance Computing, TWAREN NOC, and OF@TEIN+ community for their support.

References

[1] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1617-1634, 2014.

[2] 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, no. 2, pp. 69-74, 2008.

[3] N. McKeown, "Software-defined networking," INFOCOM keynote talk, vol. 17, no. 2, pp. 30-32, 2009.

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015.

[5] Open Networking Foundation (ONF). (2012, April 13) *Software-defined networking: The new norm for networks*. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[6] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," ACM Transactions on Internet Technology (TOIT), vol. 2, no. 2, pp. 115-150, 2002.

[7] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov and W. Snow, "ONOS: towards an open, distributed SDN OS," in Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014, pp. 1-6.

[8] O. Michel and E. Keller, "SDN in wide-area networks: A survey," in Proceedings of the IEEE Fourth International Conference on Software Defined Systems (SDS), 2017, pp. 37-42.

[9] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," IEEE Communications Magazine, vol. 51, no. 7, pp. 36-43, 2013.

[10] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang and J. Bi, "Seamless interworking of SDN and IP," ACM SIGCOMM computer communication review, Vol. 43, No. 4, pp. 475-476, 2013.

[11] P. Lin, J. Bi and H. Hu, "Internetworking with SDN using existing BGP," in Proceedings of the Ninth International Conference on Future Internet Technologies. ACM, 2014, p. 21.

[12] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012, pp. 13-18.

[13] P. W. Thai, and J. C. De Oliveira, "Decoupling BGP policy from routing with programmable reactive policy control," in Proceedings of the ACM conference on CoNEXT student workshop. ACM, 2012, pp. 47-48.

[14] Rekhter, Yakov, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," No. RFC 4271, 2005.

[15] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães, "Virtual routers as a service: the routeflow approach leveraging software-defined networks," in Proceedings of the 6th International Conference on Future Internet Technologies. ACM, 2011, pp. 34-37.
[16] J. Moy, "OSPF specification," No. RFC 1131, 1989.
[17] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, "OSHI-Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds)," in Proceedings of the IEEE Third European Workshop on Software Defined Networks (EWSDN), 2014, pp. 13-18.
[18] P. Jakma and D. Lamparter, "Introduction to the Quagga Routing Suite, " IEEE Network, vol.28, no 2, pp. 42-48, 2014.
[19] "Open vSwitch, An Open Virtual Switch," 2014. [Online]. Available: http://openvswitch.org/
[20] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of SDN in hybrid enterprise and ISP networks," in Proceedings of the Symposium on SDN Research. ACM, 2016, p. 1.
[21] P.-W. Tsai, P.-W. Cheng, H.-Y. Chou, M.-Y. Luo, and C.-S. Yang, "Toward inter-connection on OpenFlow research networks," in Proceedings of Asia-Pacific Advanced Network, vol. 36, 2013, pp. 9-16.
[22] P.-W. Tsai, P.-M. Wu, C.-T. Chen, M.-Y. Luo, and C.-S. Yang, "On the implementation of path switching over SDN-enabled network: A prototype," in Proceedings of the IEEE International Conference on Consumer Electronics-Taiwan, 2015, pp. 90-91.
[23] "Mininet: An instant virtual network on your laptop (or other PC)," 2012. [Online]. Available: http://mininet.org/
[24] "Ryu SDN Framework," 2013. [Online]. Available: https://osrg.github.io/ryu/
[25] "Oracle VM VirtualBox," 2008. [Online]. Avaliable: https://www.virtualbox.org

Hao-Ping Liu received the Bachelor's degree in the Department of Electrical Engineering from National Cheng Kung University and now is pursuing his Master's degree in the Institute of Computer and Communication Engineering at National Cheng Kung University. His research interest focuses on software-defined networking.

Pang-Wei Tsai received the Bachelor's degree in the Department of Electrical Engineering and the Master's degree in the Institute of Computer and Communication Engineering from National Cheng Kung University. His research interest is on software-defined networking, cloud computing, virtualization and network management. He is also experienced in designing the large-scale network testbed. His current research is focus on developing the future Internet testbed on TaiWan Advanced Research and Education Network.

Wu-Hsien Chang received the Bachelor's degree from the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, in 2016. He is currently pursuing the Master's degree in the Institute of Computer and Communication Engineering, National Cheng Kung University, Tainan, Taiwan. His research topic focuses on software-defined networking.

Chu-Sing Yang is a Professor of Electrical Engineering in the Institute of Computer and Communication Engineering at National Cheng Kung University, Tainan, Taiwan. He received the B.Sc. degree in Engineering Science from National Cheng Kung University in 1976 and the M.Sc. and Ph.D. degrees in Electrical Engineering from National Cheng Kung University in 1984 and 1987, respectively. He joined the faculty of the Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, as an Associate Professor in 1988. Since 1993, he has been a Professor in the Department of Computer Science and Engineering, National Sun Yat-sen University. He was the chair of the Department of Computer Science and Engineering, National Sun Yat-sen University from August 1995 to July 1999, and the director of the Computer Center, National Sun Yat-sen University from August 1998 to October 2002. He joined the faculty of the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, as a Professor in 2006. He participated in the design and deployment of Taiwan Advanced Research and Education Network and served as the deputy director of National Center for High-performance Computing, Taiwan from January 2007 to December 2008. His research interests include future classroom/meeting room, intelligent computing, network virtualization.