SDN-based Routing Application for Expansion of Electric Power Communication Networks

Wang, Ling[1] ; Jin, Xin[2] ; Song, Man Rui[1] ; Wei, Dong Xu[1] ; Sheng, Zhen[1]; Cao, Ying[3]

[1]State Grid Benxi Electric Power Supply Company, Benxi 117000, China

[2]Liaoning Medical Device Test Institute, Shenyang 110179, China

[3]Liaoning Planning and Designing Institute of Post and Telecommunication Company Limited , Shenyang 110011, China

Keywords: Software defined network; Smart Grids; Network expansion.

Abstract. With the rapid development of Internet, the network traffic explosively grows with the emergence of new network business. The existing electric power communication networks still have the traditional architecture that is the closed system highly integrated with devices and software, which cannot satisfy the development requirements of smart grids. In this paper, we combine the electric power communication networks with Software Defined Network (SDN) that includes the NOX controller in the control plane, switches in the data forwarding plane, and the OpenFlow protocol supporting the communication between controller and switches. Especially for the scenario of the network expansion, the NOX controller dynamically configures end-to-end paths, and we then demonstrate the feasibility of our design by checking the flow tables in switches. Additionally, the CPU utilization of the NOX controller and end-to-end delay are both analyzed in our experimental platform.

Introduction

The existing electric power communication networks still have the traditional architecture that is the closed system highly integrated with devices and software, which results in some serious problems. First of all, we merely rely on network devices to achieve hop-by-hop routing due to the lack of the whole network profile, and it is very difficult for us to real-time monitor the network status for the implementation of the global resource scheduling. As a result, the transmission service of power data cannot be well guaranteed, as well as the utilization of network and electric power resources is very low. Next, the distributed architecture of the power communication networks makes the unified control impossible, and thus, some functions including the fine-grained traffic analyze and access control lose efficiency. Additionally, the absence of the unified control plane also results in the low operational efficiency and huge OPEX, which cannot satisfy the development requirements of smart grids. Finally, the rapid deployment of new business is very constrained by the integration of control and data forwarding in the existing communication equipment.

Above all, it is very necessary for us to design the novel network architecture adaptive to the flexible demand of electric power communication networks [2]. Especially for the scenario of the network expansion, since the network deployment is completed by configuring devices (e.g., router or switch) independently, the managers have to manually reconfigure devices when they update a network. This approach can easily get wrong, but also consumes many manpower and material resources, especially in the large-scale data center with the complex configuration.

Therefore, to solve the aforementioned issues of the existing electric power communication networks, the Software defined network (SDN) [1] is an emerging solution. SDN decouples the control function from the data forwarding plane, meanwhile, it globally schedules and configures resources via open programmable interfaces. In this paper, we combined the electric power communication network with SDN [3, 4], with the objective of the network expansion in smart grids. More specifically, we utilize the NOX controller to dynamically provide end-to-end paths [5, 6] under the scenario of the network expansion, and then demonstrate the feasibility of our design by checking flow tables in switches. Finally, the CPU utilization of the NOX controller and end-to-end delay are both analyzed in our experimental platform.

The remaining of this paper is organized as follows. The SDN architecture, SDN-based power grid expansion, and the dynamic configuration of end-to-end paths are discussed in section 2, respectively. We demonstrate experimental results in section 3, before concluding this paper in section4.

SDN-BASED POWER GRID EXPANSION

2.1 SDN architecture

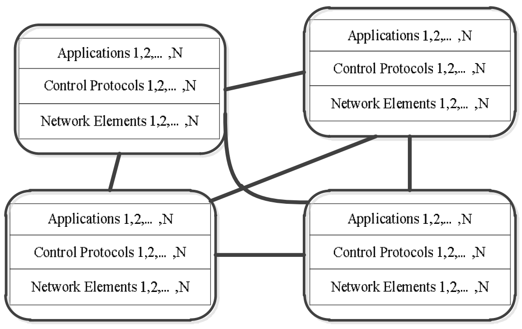


Fig. 1 Legacy network architecture

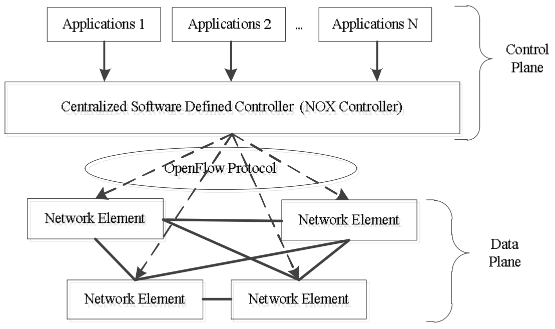


Fig. 2 SDN-based network architecture

Figure 1 shows the traditional structure of the electric power communication network that is controlled in the distributed manner since the control and data forwarding functions are integrated into the same communication equipment. The network intelligence depends on thousands of protocols. While as an emerging solution, SDN decouples the control function from the data forwarding plane, and controls the network by using the software-based method. Figure 2 is the SDN-based electric power communication network architecture including application layer, control plane and data forwarding plane. It has three features: the separation of data forwarding and control, the concentration of control logic, and the open programming interface.

2.2 Power grid expansion scenario

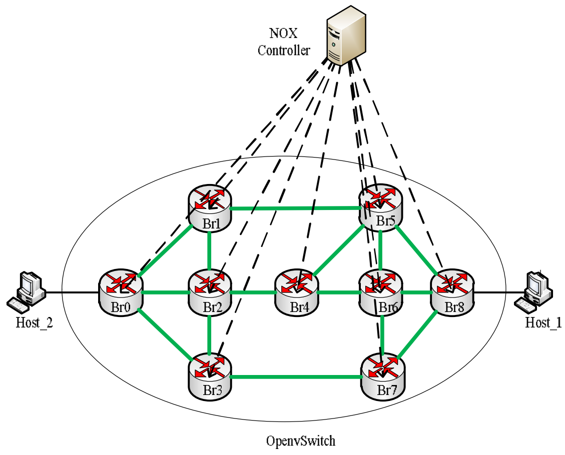


Fig. 3 Normal scenario topology

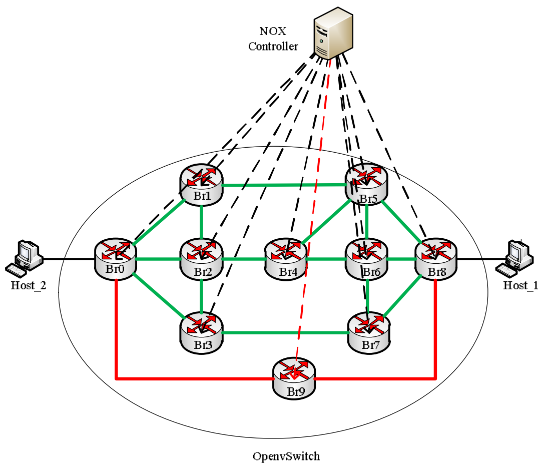


Fig. 4 Expansion scenario topology

There is a kind of scenario in smart grids: the network should be expanded by adding power distribution nodes, in order to satisfy the soaring electric power required by new residential communities. The following method is used to simulate the network expansion above. Figure 3 shows the topology of the electric power communication network before expansion, and this topology consists of a NOX controller, 9 switches and 2 hosts. Among which, Host_1 and Host_2 are source and destination nodes, respectively. We utilize OpenvSwitch to add the switch Br9 in the expanded network topology shown in Fig. 4, meanwhile, add two links Br0$\leftrightarrow$Br9, Br8$\leftrightarrow$Br9.

Owing to the feature of centralized control logic, the SDN-based network expansion can quickly provide electric power services for users. More specifically, once the switch Br9 is added into the network, NOX controller receives the 'datapath-join' event, meanwhile, it receives the link-event since two new links (Br0 ↔ Br9 and Br8 ↔ Br9) are also involved. After that, NOX dynamically updates the network topology.

2.3 Dynamic path provisioning

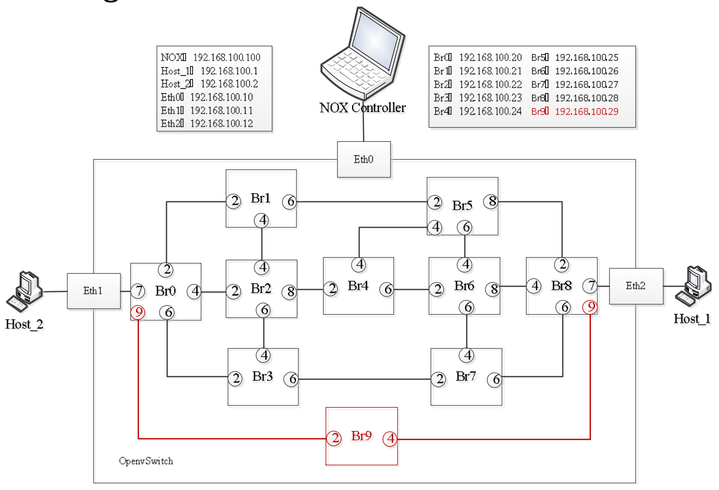
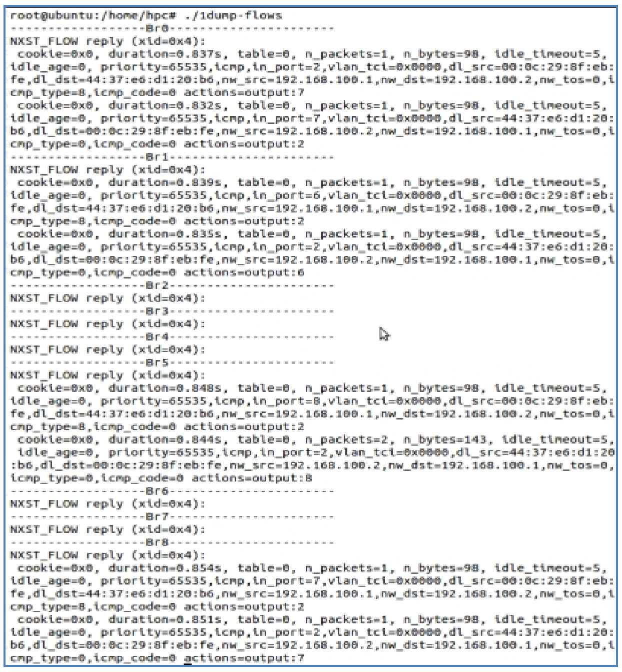


Fig. 5 Network expansion scenario based on SDN

When the packet of Host_1 arrives at Br8, the Packet_In message will be sent to the NOX controller. Next, as an example of Fig. 3, the routing application in the NOX calculates the shortest path Host_1 → Br8 → Br5 → Br1 → Br0 → Host_2 for this packet, then it sends flow tables to Br8, Br5, Br1, and Br0, respectively, by using the Flow_Mod message, in order to establish this complete path. After the network expansion, the switch Br9 and two links are added as the red part in Fig. 5. After adding the switch Br9, NOX updates the topology according to datapath-join event and link-event, and the shortest path changes into Host_1 → Br8 → Br9 → Br0 → Host_2. NOX sends flow tables to Br8, Br9, and Br0, respectively, in order to guarantee the shortest path for the packet transmission from Host_1 to Host_2.

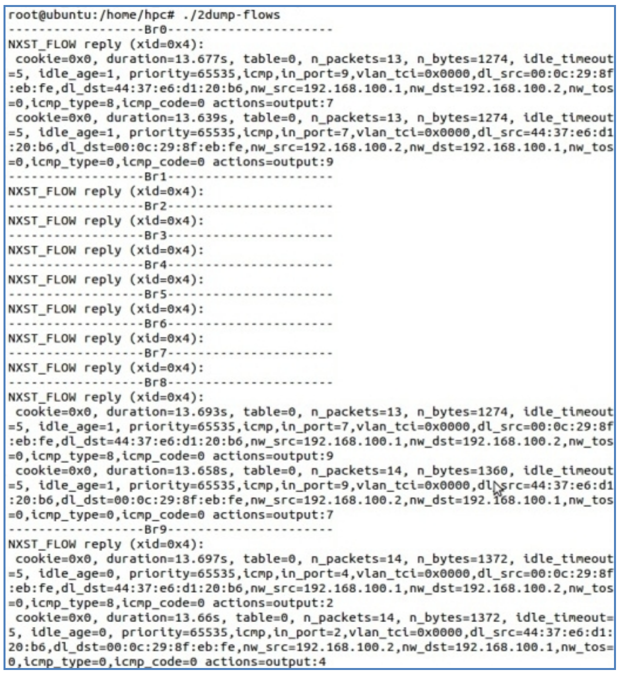
EXPERIMENTAL RESULTS AND DISCUSSIONS

3.1 Flow tables



```
root@ubuntu:/home/hpc# ./1dump-flows
------------------Br0---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.837s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=00:0c:29:8f:eb:
fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,i
cmp_type=8,icmp_code=0 actions=output:7
 cookie=0x0, duration=0.832s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=7,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:
b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,i
cmp_type=0,icmp_code=0 actions=output:2
------------------Br1---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.839s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=6,vlan_tci=0x0000,dl_src=00:0c:29:8f:eb:
fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,i
cmp_type=8,icmp_code=0 actions=output:2
 cookie=0x0, duration=0.835s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:
b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,i
cmp_type=0,icmp_code=0 actions=output:6
------------------Br2---------------------
NXST_FLOW reply (xid=0x4):
------------------Br3---------------------
NXST_FLOW reply (xid=0x4):
------------------Br4---------------------
NXST_FLOW reply (xid=0x4):
------------------Br5---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.848s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=8,vlan_tci=0x0000,dl_src=00:0c:29:8f:eb:
fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,i
cmp_type=8,icmp_code=0 actions=output:2
 cookie=0x0, duration=0.844s, table=0, n_packets=2, n_bytes=143, idle_timeout=5,
 idle_age=0, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:20
:b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,
icmp_type=0,icmp_code=0 actions=output:8
------------------Br6---------------------
NXST_FLOW reply (xid=0x4):
------------------Br7---------------------
NXST_FLOW reply (xid=0x4):
------------------Br8---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.854s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=7,vlan_tci=0x0000,dl_src=00:0c:29:8f:eb:
fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,i
cmp_type=8,icmp_code=0 actions=output:2
 cookie=0x0, duration=0.851s, table=0, n_packets=1, n_bytes=98, idle_timeout=5,
idle_age=0, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:
b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,i
cmp_type=0,icmp_code=0 actions=output:7
```

Fig. 6 Flow tables of switches before network expansion



```
root@ubuntu:/home/hpc# ./2dump-flows
------------------Br0---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=13.677s, table=0, n_packets=13, n_bytes=1274, idle_timeout
=5, idle_age=1, priority=65535,icmp,in_port=9,vlan_tci=0x0000,dl_src=00:0c:29:8f
:eb:fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos
=0,icmp_type=8,icmp_code=0 actions=output:7
 cookie=0x0, duration=13.639s, table=0, n_packets=13, n_bytes=1274, idle_timeout
=5, idle_age=1, priority=65535,icmp,in_port=7,vlan_tci=0x0000,dl_src=44:37:e6:d1
:20:b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos
=0,icmp_type=0,icmp_code=0 actions=output:9
------------------Br1---------------------
NXST_FLOW reply (xid=0x4):
------------------Br2---------------------
NXST_FLOW reply (xid=0x4):
------------------Br3---------------------
NXST_FLOW reply (xid=0x4):
------------------Br4---------------------
NXST_FLOW reply (xid=0x4):
------------------Br5---------------------
NXST_FLOW reply (xid=0x4):
------------------Br6---------------------
NXST_FLOW reply (xid=0x4):
------------------Br7---------------------
NXST_FLOW reply (xid=0x4):
------------------Br8---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=13.693s, table=0, n_packets=13, n_bytes=1274, idle_timeout
=5, idle_age=1, priority=65535,icmp,in_port=7,vlan_tci=0x0000,dl_src=00:0c:29:8f
:eb:fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos
=0,icmp_type=8,icmp_code=0 actions=output:9
 cookie=0x0, duration=13.658s, table=0, n_packets=14, n_bytes=1360, idle_timeout
=5, idle_age=1, priority=65535,icmp,in_port=9,vlan_tci=0x0000,dl_src=44:37:e6:d1
:20:b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos
=0,icmp_type=0,icmp_code=0 actions=output:7
------------------Br9---------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=13.697s, table=0, n_packets=14, n_bytes=1372, idle_timeout
=5, idle_age=0, priority=65535,icmp,in_port=4,vlan_tci=0x0000,dl_src=00:0c:29:8f
:eb:fe,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos
=0,icmp_type=8,icmp_code=0 actions=output:2
 cookie=0x0, duration=13.66s, table=0, n_packets=14, n_bytes=1372, idle_timeout=
5, idle_age=0, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:
20:b6,dl_dst=00:0c:29:8f:eb:fe,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=
0,icmp_type=0,icmp_code=0 actions=output:4
```

Fig. 7 Flow tables of switches after network expansion

Figure 6 shows the results of flow tables before network expansion. We can see that, the shortest path is Host_1 → Br8 → Br5 → Br1 → Br0 → Host_2. Figure 7 shows the results of flow tables after network expansion, and the shortest path changes into Host_1 → Br8 → Br9 → Br0 → Host_2. In addition, the results in Fig. 7 demonstrate that NOX can dynamically update the topology and re-compute the shortest path for the packet, so that the high-quality electric power service is guaranteed after network expansion.

3.2 CPU utilization of NOX

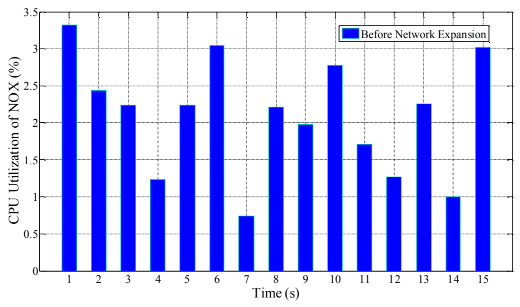


Fig. 8 CPU utilization of NOX before network expansion

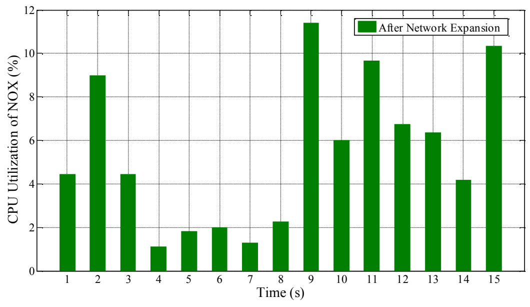


Fig. 9 CPU utilization of NOX after network expansion

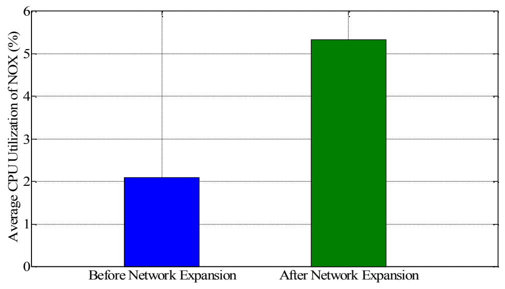


Fig. 10 Average CPU utilization of NOX

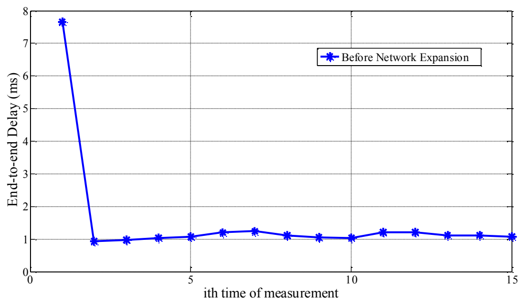


Fig. 11 End-to-end delay before network expansion

Figures 8 and 9 demonstrate the CPU utilization of the NOX controller before and after network expansion, respectively. The horizontal axis is the running time, i.e., one sampling one second, and the vertical axis records the CPU utilization. Figure 10 compares the average CPU utilization before and after network expansion. The CPU utilization of the NOX controller under the network expansion scenario is higher than that before network expansion. This is because the topology will be updated and the new shortest path should be determined by the NOX controller.

3.3 End-to-end delay

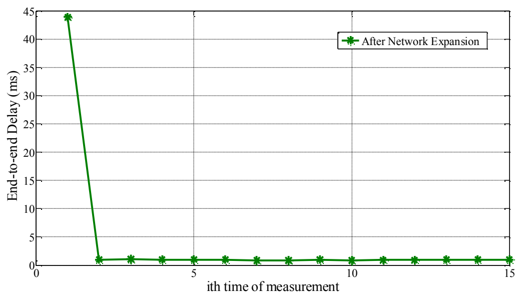


Fig. 12 End-to-end delay after network expansion

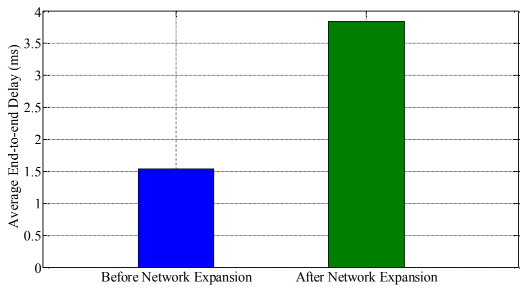


Fig. 13 Average end-to-end delay

Figures 11 and 12 show the results of end-to-end delay before and after network expansion, respectively. Among which, the horizontal axis is the packet arrival sequence, while the vertical axis records the end-to-end delay of the packet transmission from Host_1 to Host_2. The reason of the first packet has a long delay is that when the first packet arrives at the network, no flow table has been established, so we need the complex process of flow-table establishment. More specifically, we have to send Packet_In message to the NOX controller for the purpose of calculating the shortest path, and then NOX sends the Flow_Mod message to switches for the establishment of flow tables. Figure 13 compares the end-to-end delay before and after network expansion. The average delay increases after the network expansion because NOX consumes time to dynamically update the topology and resend flow tables.

CONCLUSION

The OpenFlow-based SDN had the features of centralized control and distributed data forwarding, which made it enable to globally control the network resource, and highly increase the operational efficiency. As for the network expansion scenario, NOX could dynamically provide end-to-end paths. The experimental results demonstrate that the combination of SDN and the electric power communication network was feasible and convenient for the new service deployment. Therefore, the OpenFlow-based SDN has wide application prospect in the electric power communication network.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, et al. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[2] Jianchao Zhang, Boon-Chong Seet, Tek-Tjing Lie et al. Opportunities for Software-Defined Networking in Smart Grid. 2013 9th International Conference on Information, Communications and Signal Processing (ICICS), Tai nan, Dec. 2013, pp.1-5.

[3]. J. Rubio-Loyola et al. Scalable Service Deployment on Software-defined Networks. IEEE Communications Magazine, 2011, 49(12): 84-93.

[4]. M. Mendonca, K. Obraczka, and T. Turlett. The Case for Software-defined Networking in Heterogeneous Networked Environments. Proc. of ACM CoNEXT workshop, Nice, France, December 2012, pp.59-60.

[5]. L. Lei, Y. C. Hyeon, C. Ramon, et al. Demonstration of a Dynamic Transparent Optical Network Employing Flexible Transmitters/Receivers Controlled by an OpenFlow–Stateless PCE Integrated Control Plane. IEEE/OSA Journal of Optical Communications and Networking, 2013, 5(10): 66-75.

[6]. Neda Cvijetic, Akihiro Tanaka, Philip N. Ji, et al. SDN and OpenFlow for Dynamic Flex-Grid Optical Access and Aggregation Networks. IEEE/OSA Journal of Lightwave Technology, 2014, 32(4): 864-870.