2013 IEEE 37th Annual Computer Software and Applications Conference

OpenFlow Network Visualization Software with Flow Control Interface

Yasuhiro Watashiba*, Seiichiro Hirabara*, Susumu Date†, Hirotake Abe‡,
Kohei Ichikawa§, Yoshiyuki Kido†, Shinji Shimojo† and Haruo Takemura†

*Graduate School of Information Science and Technology, Osaka University, Japan
Email: {watashiba-y@, hirabara.seiichiro@ais.}cmc.osaka-u.ac.jp
†Cybermedia Center, Osaka University, Japan
Email: {date, kido, shimojo, takemura}@cmc.osaka-u.ac.jp
‡Faculty of Engineering, Information and Systems, University of Tsukuba, Japan
Email: habe@cs.tsukuba.ac.jp
§Graduate School of Information Science, Nara Institute of Science and Technology, Japan
Email: ichikawa@is.naist.jp

***Abstract*—Recently, the concept of Software-Defined Network (SDN), which allows us to administer and configure a network in a centralized and software-programming manner, has gathered network engineers' and researchers' attention rapidly. In particular, the expectation and concern to OpenFlow as an implementation of the SDN is remarkable. As a result, research activities, which include prototyping, implementation, demonstration and experiments, conducted over OpenFlow networks have been a worldwide tendency. In such research activities, however, the difficulty in understanding network topology, traffic amount and an actual path of a network flow on the OpenFlow network, and the intricacies in debugging software designed for OpenFlow are serious problems in the development process of OpenFlow controller. This research aims to realize a visualization software that facilitates researchers to perform OpenFlow controller development and demonstration experiments performed on an actual OpenFlow network. In this paper, the authors summarize the achievement of their research work in progress as well as the future direction.**

I. Introduction

Recently, Software-Defined Network [1], a newly emerged concept of network architecture has gathered much concern and expectation from network engineers and researchers. SDN decomposes networking functions equipped on ordinary network switches and routers to data forwarding function and control function. In SDN, the control function performed on each switch is aggregated to a software application as a SDN controller. This concept enables us to dynamically configure and control the entire network through the controller in a centralized manner. Also, it is expected as a network virtualization technology that is able to provide multiple isolated virtualized experimental networks simultaneously.

Today, in particular, OpenFlow has become a promising implementation technology of the SDN concept. In general, an OpenFlow network is usually composed of an OpenFlow controller responsible for the control functions for handling network packets and multiple OpenFlow-capable network switches. OpenFlow protocol provides an open standard to manage flow-tables in multi-vendor network facilities [2]. These switches and routers communicate with the controller through the OpenFlow protocol. Making use of the protocol, a network administrator can partition traffic into production and research flows.

From such expectation and concern to the SDN and OpenFlow, many research activities and experiments using OpenFlow networks have been actively performed in these days [3], [4]. In general, the operation of an OpenFlow network requires the development of an OpenFlow controller, so that the OpenFlow controller as a software program can handle flow-tables on OpenFlow-enabled network switches and routers [5]. For the reason, several kinds of development frameworks which facilitate us to code OpenFlow controller, such as POX [6], Trema [7] and Floodlight [8] have been available.

Despite of the existence of such frameworks, the developer tends to have the following two difficulties. The first difficulty lies in understanding the topology of the entire OpenFlow network, actual paths of the network flows, traffic amount on each link of the network and so forth. The second difficulty is that the developer would be forced to go through try-and-error process, in other words, to repeat the process of modifying a program as a controller and then confirming the behavior of the program, accompanied with the restart of the whole OpenFlow network.

In this paper, we report a research work in progress towards OpenFlow network visualization software, which can facilitate us to efficiently develop an OpenFlow controller. The rest of the paper is structured as follows. In section II, we review the process of OpenFlow controller development and consider the issues to go up the development efficiency. In section III, we show the OpenFlow network visualization software which allows us to view the topology of our target OpenFlow network and operate flows on it with ease. We conclude the paper in section IV.

II. Problem in OpenFlow Controller Development

In OpenFlow, data plane responsible for packet forwarding function is deployed on OpenFlow network devices such as

0730-3157/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSAC.2013.81

XML-RPC(Remote Procedure Call):
1. 是一個可在不同的操作系統上運作(Unix, Windows and Macintosh)
2. 可使用不同程式語言編譯(Perl, Java, Python, C/C++)
3. 遠程過程中使用HTTP作為傳輸和XML(HTML)作為編碼

switches and routers. On the other hand, control plane responsible for routing decision is deployed on OpenFlow controller. This separation of network functions allows each OpenFlow switch to handle packets on a per-flow basis. Furthermore, this per-flow controllability of packets enables the fine-grained slicing of network resources, which results in multiple isolated virtualized networks.

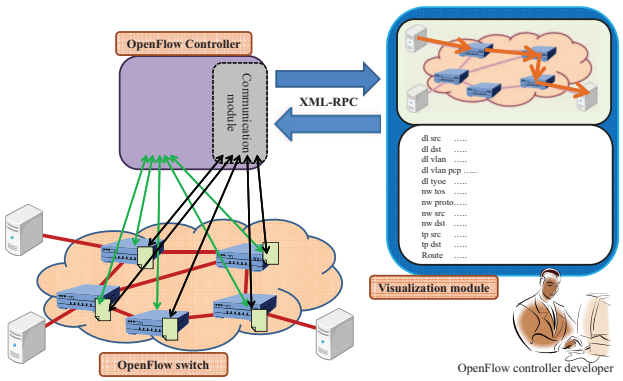
The development of an OpenFlow controller necessitates fair amount of careful consideration and error-prone operation due to the controllability of network packets on a per-flow basis. In most cases, the development of OpenFlow controller involves the following steps.

1) Placement and deployment of data plane in OpenFlow network.
2) Designing and coding of OpenFlow controller.
3) Experimental operation of OpenFlow network.
4) Verification of OpenFlow controller behavior.

Usually, these steps need to be repeated for confirming the precise behavior of the developed controller. Importantly, the developer has to understand how flow isolation and resource slicing are performed in data plane in addition to the physical topology of the target OpenFlow network. Furthermore, particularity in the case that a group of developers collaborate, they have to have a common view of virtualized space and physical space of the target OpenFlow network. Moreover, the developer is required to restart the software program as the controller many times every time he or she modifies the program. These factors lower the efficiency in development.

III. Proposal and Status Report of our Work in Progress

For the consideration above, we have been prototyping visualization software that can give OpenFlow controller developers an intuitive understanding of how network flows are formed and what the topology of a physical OpenFlow network is like. Also, the visualization software is being developed with a flow control interface that allows the developers to temporarily add and remove flow-entries directly to OpenFlow switches via OpenFlow controller, without having OpenFlow controller restarted.



主要功能:
1. OpenFlow switch:
   -負責與 OpenFlow controller 連接並且將封包轉送到目的端
   -聽命於 OpenFlow controller 行動做事
2. OpenFlow controller:
   -管理每個 OpenFlow switch 封包該如何走
   -透過網頁(XML-RPC)操作方式來呈現

Figure 1 overviews the architecture of our visualization software. The visualization software under our development is composed of visualization module, communication module built in an OpenFlow controller, and OpenFlow switches. The visualization module is in charge of visualizing a topology of the OpenFlow network as a graph and then superimposing network flow information on the graph. Also, through the visualization module, the developers can add and remove flow entries to OpenFlow switches. The communication module built in OpenFlow controller is responsible for obtaining the information on network from switches and mediating the interaction between the developer and OpenFlow switches. For the communication between the communication module and OpenFlow switches, OpenFlow is used. For the interaction with the visualization module, XML-RPC [9], [10] has been adopted in this research. XML-RPC is an easy and popular protocol to make a Remote Procedure Call (RPC) over the network. In this implementation, the communications between the visualization module and the communication module in an OpenFlow controller are performed by XML-RPC. For the development of OpenFlow controller, we have adopted Trema, which is a programming framework for OpenFlow controller, focusing on the characteristics of enabling the development of OpenFlow controller in Ruby and C.

A. Visualization module

The visualization module has two functionalities. The first functionality is to visualize the topology of the target OpenFlow network and superimpose the information on network flows on it. The information on network flows include traffic amount on each link between OpenFlow switches composing the OpenFlow network, a list of network flows, traffic amount of each link. For visualization of network, we have adopted Jung 2.0.1 (Java Universal Network/Graph Framework) [11]. The second functionality is to allow the developer to directly describe flow entry parameters. Figure 2 shows the snapshot of the visualization software under development.

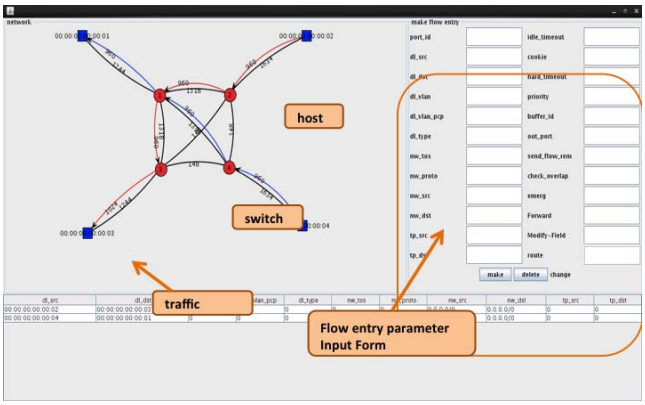


Fig. 2. Snapshot of visualization software under development.

B. Communication module

The communication module makes the topology graph of Flow network by utilizing LLDP (Link Layer Discovocol), a protocol for information exchange between devices. The communication module first instructs

each OpenFlow switch to send LLDP packets to all neighbors. A LLDP packet contains the ID of the sender switch and the ID of the port that it is sent through. On receiving the LLDP packet from a neighbor switch, OpenFlow switches report its arrival to the communication module. With this mechanism, the communication module can obtain the information on all links and then build the topology graph of the OpenFlow network which OpenFlow switches form. Furthermore, to understand the connections between OpenFlow switches and hosts, the communication module works as follows. In the event that a new host is connected to an OpenFlow switch and sends out any packet, the event is informed to the communication module built in the OpenFlow controller. Then, the communication module extracts source MAC address and IP address contained in the packet. If their address is unknown to the communication module, it adds the corresponding connection between the OpenFlow switch and the new host to the graph obtained by the LLDP-based mechanism above.

To obtain the information on network flows on the OpenFlow network, the communication module analyzes flow entry information retrieved from each OpenFlow switch. Figure 3 shows an example of flow entry information retrieved from an OpenFlow switch. As the figure shows, the flow entry information retrieved from an OpenFlow switch contains only which port to send out for a series of packets that matches a rule set in advance. Furthermore, the information is fragmentary and thus the developer has difficulty in understanding the entire network flows on the target OpenFlow Network. In the communication module, "match", "actions", and "packet_count" are focused. In order to understand the network flows on the OpenFlow network, the communication module obtains these information from each switch every 1 second. Also, the traffic amount of each network flow is estimated from the comparison of "byte_count" flowed on a specific port between the information obtained every 1 second.

The communication module also has the ability to modify the actual path of a specified flow. This is conducted by adding new flow entries which override the flow entries of the specified flow. Note that the modified path will be reverted some time later because the original OpenFlow controller will refresh the flow entries in the event of timeout.

```
#<Trema::FlowStatsReply:0xb765928c>
actions: #<Trema::SendOutPort:0xb765937c>
byte_count: 38336
cookie: 767300786513248268
duration_nsec: 924000000
duration_sec: 21
hard_timeout: 0
idle_timeout: 0
length: 96
match: wildcards = 0x3820f3(in_port|dl_type|
dl_vlan|dl_vlan_pcp|nw_proto|nw_tos|nw_src(32)
|nw_dst(32)|tp_src|tp_dst), in_port = 0,
dl_src = 00:00:00:00:00:02, dl_dst = 00:00:00:00:00:01,
dl_vlan = 0, dl_vlan_pcp = 0, dl_type = 0,
nw_tos = 0, nw_proto = 0, nw_src = 0.0.0.0/0,
nw_dst = 0.0.0.0/0, tp_src = 0, tp_dst = 0
packet_count: 599
priority: 65530
table_id: 0
```

Fig. 3. Flow entry information retrieved from OpenFlow switch.

C. OpenFlow switch

As of writing this paper, we do not have enough OpenFlow switches to make an OpenFlow network. In this research, Open vSwitch [12], [13] has been used for emulation of OpenFlow switch. Open vSwitch is a software implementation of an OpenFlow switch intended to run as a virtual switch in virtualized environments. Open vSwitch can operate both as a software switch running within the hypervisor, and as the control stack for switching silicon. It has been ported to multiple virtualization platforms and switching chipsets [13]. In this research, an OpenFlow network has been constructed with Open vSwitches in a Linux machine.

IV. Conclusion

In this paper, our research work in progress toward OpenFlow network visualization software was reported. We have focused on the difficulties in developing an OpenFlow controller and thus started the development of OpenFlow network visualization software for the purpose of facilitating the developer's coding. At the time of writing this paper, the software is still under development. The authors would like to improve the prototyped visualization software obtaining the feedback from actual developers and engineers working around OpenFlow network.

Acknowledgment

This research was partly supported by the collaborative research of National Institute of Information and Communications Technology (NICT) and Osaka University (Research on high functional network platform technology for large-scale distributed computing).

References

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," ONF White Paper, April 12, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", ACM SIGCOMM Computer Communication Review, Vol. 38, No. 2, pp.69-74, Apr. 2008.

[3] GENI, http://www.geni.net/.

[4] JGN-X, http://www.jgn.nict.go.jp/english/.

[5] T. Furuichi, S. Date, H. Yamanaka, K. Ichikawa, H. Abe, H. Takemura, and E. Kawai, "A prototype of network failure avoidance functionality for SAGE using OpenFlow", Proceedings of 2012 IEEE 36th International Conference on Computer Software and Applications Workshops (The Sixth Middleware Architecture in the Internet (MidArch 2012)), pp.88-93, Jul. 2012.

[6] POX, http://www.noxrepo.org/pox/about-pox/.

[7] Trema, http://trema.github.io/trema/.

[8] Floodlight OpenFlow Controller - Project Floodlight, http://www.projectfloodlight.org/floodlight/.

[9] S. S. Laurent, E. Dumbill, and J. Johnston, "Programming Web Services with XML-RPC", O'Reilly Media, Incorporated, 2001.

[10] XML-RPC.Com, http://xmlrpc.scripting.com/.

[11] JUNG - Java Universal Network/Graph Framework, http://jung.sourceforge.net/.

[12] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado and S. Shenker, "Extending Networking into the Virtualization Layer," HotNets-VIII, 2009.

[13] Open vSwitch, http://openvswitch.org/.