DEPENDENCY ANALYSIS SYSTEM TO NARROW DOWN MISSETTINGS IN SDN CONSTRUCTION EXERCISES USING TREMA

Hlwam Maint Htet

Faculty of Computer Systems and Technologies, University of Computer Studies, Yangon,
Shwe Pyi Thar Township, Yangon, 11411, Myanmar
hlwammainthtet@ucsy.edu.mm
https://ucsy.edu.mm

Abstract

Software defined networking (SDN) has been popular for research and innovation. Universities and research labs are the basic points for innovation because innovation by academia and research organizations can accelerate the rate of change in industries. SDN construction exercises have been developed in e-Learning. When performing network construction exercises, novice learners cannot understand the behavior of their network and fail to satisfy the requirements for the network reachability of communication data. In this system, learners construct SDN with Trema as exercises. A communication protocol OpenFlow is used for communication between controllers and switches. Here, some learners cannot find their bugs from their settings due to the following reasons such as (1) ping cannot find delivery routes including switches, (2) switches have no function to log rules used for choosing output ports for packets and (3) Trema cannot find execution statements used for setting rules to switches. To satisfy these problems, the system will provide clues for learners in visual way so that they can narrow down executed statements that cause incorrect communication.

***Keywords*: SDN, network construction exercises, Trema, OpenFlow.**

1. Introduction

Virtualization technology has been developed in momentum and various uses of this technology in industries and academia has become popular because of its benefits in every area as the user can construct his/her own networks on every time/everywhere without any constraints. Network virtualization technology enables researchers to share the testbed simultaneously [7]. Because of the network traffic has increased with the development of cloud services, network infrastructures have been optimized by controlling network behavior using software defined networking (SDN) [4]. With the wide use of SDN, network construction exercises using SDN have been developed in some universities and research organizations. In this work, we construct networks exercises in universities with the use of Trema, [1] a framework for developing controller programs. Some novice learners cannot understand the behavior of their networks and cannot find their bugs from their settings. We implement functions to collect packets form virtual networks, to collect executed statements in controller programs, to visualize packet location and executed statements in chronological order, and to algorithms to analyze dependency between communication routes and flow tables and executed controllers. The system outputs three clues for learners to narrow down missettings in SDN construction exercises.

2. Related Work

This section describes the related works for narrowing down missettings in SDN construction exercises. In [2], authors describe the method of students' learning state by generating colour-coded time chart of learning status in which students are experiencing learning setbacks. "A fast method of verifying network routing with back-trace header space analysis", Toshio Tonouchi Satoshi Yamazaki, 2015 IEEE proposed a new method of fast verification that is called back-trace header space analysis (B-HSA), a variation of the HAS in [3]. It provides a proof that the proposed method can correctly verify reachability and isolation. Here, it cannot detect a loop route, which is a critical failure in the network and it does not calculate the header-spaces at outgoing ports.

"Determining Learning Status in SDN Construction Exercises", Takashi Yokoyama, Hisayoshi Kunimune, Masaaki Niimura, Shinshu University Japan, E-Learn 2016 - Washington, DC, United States, November 14-16, 2016, focus on detecting students who experience learning setbacks. The purpose of this paper is to identify

students experiencing setbacks in writing source code for an OpenFlow controller in real time [2]. Proposed a method for identifying struggling students by coloring on time chart about learning states.

A System for Generating Hints on Network Construction Exercises for Beginners", Yuichiro Tateiwa, Nagoya University, Japan, ICCE 2016 IEEE in [4]. It is hard to logically discuss how to derived fault regions, because conventional exercise problems are described in natural language. Therefore, formal representation that defined the composition of exercise problems is needed. Developed a hint generation system that displays information regarding behavior errors as hints to learners by displaying static images of the behavior of both networks in parallel and messages describing the errors.

3. Background and Methodology

This section describes how the system was proposed and which methods are used in this system.

3.1. *SDN and Openflow Controller*

Progress in virtualization technology has been rapid due to the spread of cloud services. Therefore, requests for network are changing and network configuration must change dynamically. It is difficult for legacy networks to solve these problems. For this reason, dynamic and fast techniques for network construction are required. SDN is one of the techniques that meet these criteria. It has two defining characteristics: First, it separates the control plane (where the decisions of handling traffic are made) from the data plane (where the actual forwarding of traffic takes place). Second, an SDN consolidates the control plane so that a single software control program controls multiple data-plane elements. OpenFlow is a Layer 2 communication protocol that gives access to the forwarding plane of a network switch or router over the network. Open means that the interface for externally controlling the switches is open, enabling anyone to participate in modifying the switch functions. Flow means that the control is based on a flow which can be arbitrarily defined. In general, an OpenFlow controller is developed using the following steps:
1. Writing source code for the OpenFlow controller
2. Executing the controller
3. Executing commands (equivalent to debugging)
4. Modifying the source code and repeating steps 2 and 3 as needed

3.2. *Trema*

Trema [1] is a framework to develop Controller Programs and it was used by researchers, network operators, or service creators to develop their own OpenFlow controllers with multiple control modules. It includes an integrated network emulator, which consists of pseudo hosts and virtual switches, so that users can easily test a new controller with this network emulator and then seamlessly deploy to production environment. For easier debugging in a distributed system called network, Trema provides an integrated debugging environment that collects state information from all parts of the system. Trema serves as a platform for building and deploying OpenFlow controllers. These controllers manage network devices (such as switches and routers) that support the OpenFlow protocol, enabling centralized control and management of network traffic. It has three features like (1) controller development, (2) plugin architecture, and (3) testing and simulation.
Trema is primarily written in Ruby, which makes it accessible for developers familiar with Ruby programming. It provides Ruby bindings and APIs for interacting with OpenFlow-enabled devices. As an open-source project, Trema benefits from community contributions and has a community-driven development model. Users can leverage community forums, GitHub repositories, and documentation to learn and contribute to the project.
Overall, Trema plays a crucial role in the development and deployment of OpenFlow controllers, offering a flexible and extensible framework for building software-defined networking solutions. It empowers developers to implement customized network control logic and leverage the benefits of SDN in modern network architectures

4. System Implementation

The purpose of this paper is to give clues for learners when they are facing errors during network construction exercises. We added a standard output function (prefix, line number, and clock) to all statements of the controller program. Figure (1) is a block diagram of the proposed system. In the proposed system, the network configuration information of the exercise problem and the controller program are first received from the user. In such a case, the data collection function starts packet capture and Trema, and logs the communication test performed by the user. When the user finishes the communication test and makes a termination request, the data collection function terminates packet capture and Trema, obtains packets, OpenFlow messages, and execution history, and saves them in the log DB as packet information, OpenFlow information, and executable statement information to do. When the data collection function ends, the time series flow table reproduction function, route selection imitation function, and transmission route estimation function are executed in order, and the time series flow table, route

selection history, and transmission route information are created and saved in the log DB. There is a clue creation function as a means of obtaining necessary information from the log DB, and this function is obtained from the log DB and returns the above three clues to the user.

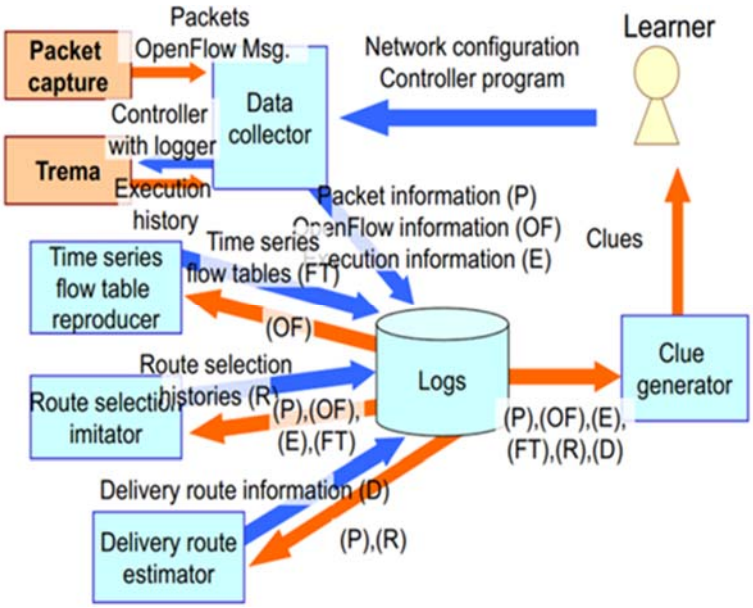

Figure 1. System Overview

4.1. *SDN Construction Exercises*

The flow of SDN construction exercises is shown in Figure 2. In this exercise problem, the network configuration is given. The goal of the exercise is to achieve the communication test successfully. The learners precede the exercise using Trema according to the procedure. First, create a controller program. Next, set the virtual network environment using a shell script. Then, execute the controller according to the controller executing procedure and perform a communication test**.**

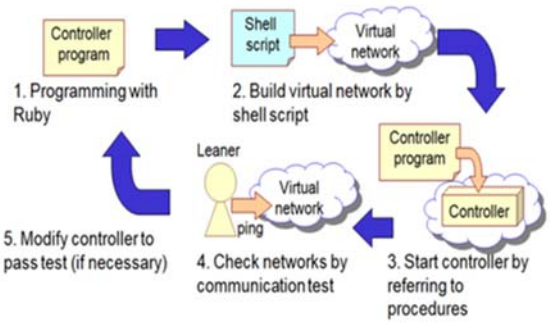

Figure 2. Flow of the exercise

4.2 *Network Topology*

The sample of network construction exercise topology is shown in Figure 3. It includes 3 hosts, 3 virtual switches. Which is used to test the network construction exercises for learners.

Figure 3. Network topology with three virtual switches

4.3 Data Collector

As in Figure 4, we create a data collection function with Ruby and C to obtain the execution history. In this function, the input is created to obtain the execution history. Execution history collection programs are regulated by prefix line numbers. As an example, if the original executable statement is a packetin method definition statement, add a description to output the argument in addition to the prefix, line number, and time as shown below. It was regulated by prefix line number. The data collection function also associates PacketIn and controller executable statements. The controller does not always process PacketIn in the order received.

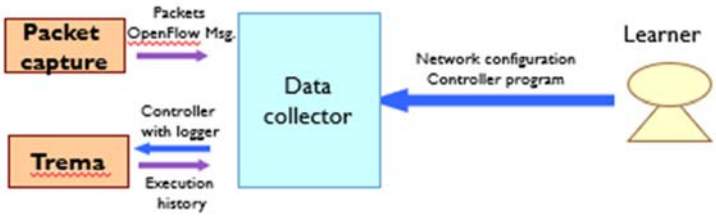

Figure 4. Collecting Data

4.4 Delivery Route Estimator

Figure 5. describes the transmission path estimation function. This function estimates the transmission route from the route selection history. The destination device will be terminated. This estimates the transmission path. IT estimate delivery routes based on route selection histories. Firstly, find route selection history "A" whose sending device is a host then it finds the route selection history "B" that satisfies the following conditions;
Reception device of A == Sending device of B
Sending device of A == Reception device of B
Packet sent from A == Packet received by B
Reception time of A < Reception time of B
If B's reception device is not a host, then assign B to A and go back to step 2.

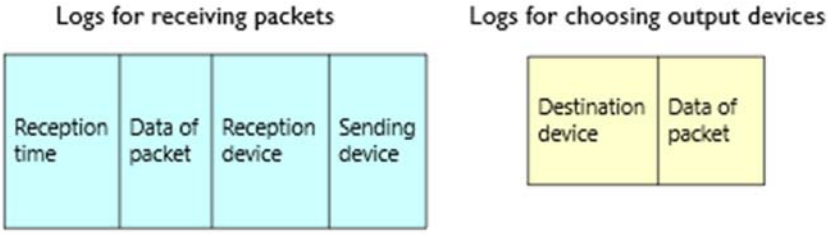

Figure 5. Delivery Route Estimation Function

5. Experiment

In this system, we set three communication networks as sample exercises. The achievement conditions are do ping communications between all pairs of hosts and all hosts in each pair send three ICMP echo requests. The evaluation environment includes Host OS: Windows 10, Host CPU: Intel(R) Core (TM) i7-6500U CPU @ 2.59GHz, Virtual Machine: VMware Workstation 15 Player, Guest OS: Ubuntu 16.04LTS and Guest memory assignment is 1GB. The visualizer is implemented with Ruby and C language, while the Plant UML converts the description to graphics. It might make it possible for us to achieve our goals. The analysis results are shown in Table 1.

Network	Average RTT (ms)		Amount of Data (KB)		
	Proposed Controller Program	Ordinary Controller Program	Packet Information	Openflow Information	Executed Statement Information
A	4.5	10	18	20	2
B	10.6	12.3	22	25	3
C	20	12.9	25	30	4

Table 1. Analysis result.

6. Conclusion and Future Work

In this system, we have analyzed that the dependency between the communication activities in SDN construction exercises to narrow down missettings and to help learners while doing SDN construction exercises. We used OpenFlow controller Trema. In our work, we proposed a system for analyzing dependency in communication activity in SDN construction exercises using Trema. Learners cannot identify missettings in SDN controllers due to many reasons such as ping cannot find delivery routes including switches, switches have no function to log rules. For learners who have difficulty narrowing down errors, the proposed system can satisfy these problems and can obtain three clues from the dependency analysis that help learners to identify rules, to narrow down switches and to narrow down executed statements that cause incorrect communication. We also evaluate the system performance and effectiveness for students' debugging.

Conflict of Interest

Declared conflicts of interest do not exist for the authors. There are no financial conflicts to disclose,

Acknowledgments

I would like to thank Dr. Amy Tun, Faculty of Computer Systems and Technologies from University of Computer Studies, Yangon, Myanmar and Professor Dr. Yuichiro Tateiwa, Nagoya Institute of Technology, for their kind guidance and supervising to complete and publish this research work.

References

[1] "Trema: A Brief Introduction and Tutorial", Shuji Ishii, Eiji Kawai, APAN 32nd Future Internet Testbed Workshop, 2011.

[2] Takashi Yokoyama, Hisayoshi Kunimune, Masaaki Nimura, "Determining learning status in SDN construction exercises", E-learn 2016, Washington, DC, United States, November 14-16, 2016

[3] "A fast method of verifying network routing with back-trace header space analysis", Toshio Tonouchi Satoshi Yamazaki, 2015 IEEE.

[4] "A System for Generating Hints on Network Construction Exercises for Beginners", Yuichiro Tateiwa, Nagoya University, Japan, ICCE 2016 IEEE.

[5] Alexander Shallimov, Dmitry Zulkov, Daria Zimarina, Vasily Pashikov, Ruslan Smeliansky, Advanced Study of SDN/OpenFlow Controllers, CER-SCER'13 Moscow, Russia

[6] "Proposal of an event visualization system for debugging in software-defined networking exercises using Trema", Yuichiro Tateiwa, Nagoya Institute of Technology, Japan, ICCE 2020 IEEE.

[7] "Programmable Network Using OpenFlow for Network Researches and Experiments", HIDEyuki Shimonishi, Yasuhito Takamiya, Yasunobu Chiba, Kazushi Sugyo, Youichi Hatano, Kentaro Sonoda, Kazuya Suzuki, Daisuke Kotani, and Ippei Akiyoshi, ICMU 2012.

[8] "Automatic Test Packet Generation", Hongyi Zeng, Peyman Kazemian, George Varghese§, Nick McKeown, IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 22, 2014.

Authors Profile

In 2013, **Hlwam Maint Htet**, earned her Master of Computer Science (M.C.Sc) degree with credits from the University of Computer Studies, Pakokku, Myanmar in 2013. She is currently pursuing her Ph.D at the University of Computer Studies, Yangon, Myanmar. She is currently working as a Staff Officer in the Internal Revenue Department, Ministry of Planning and Finance, Myanmar and working as a Chief System Manager of IT Infrastructure and Cyber Security Section. Her other research interests are Networking, Virtualization Technology, Cyber Security. She can be contacted at email: hlwammainthter@ucsy.edu.mm.