A Database Oriented Management for Asynchronous and Consistent Reconfiguration in Software-Defined Networks

Yuki Kawai†, Yasuhiro Sato*, Shingo Ata†, Dijiang Huang‡, Deep Medhi§, and Ikuo Oka†
†Osaka City University, Japan
*Japan Coast Guard Academy, Japan
‡Arizona State University, USA
§University of Missouri–Kansas City, USA

Abstract—Software-Defined Networking (SDN) is a new approach to manage the whole network flexibly by decoupling the control plane and the forwarding plane. While forwarding elements can be managed by a unified control, complexity arisen from the network size and scalability regarding the increase of the control traffic are notable problems. To deal with events of network reconfiguration that occur asynchronously and change frequently with intervals shorter than hours, a controller has to continue to asynchronously update the configuration of the whole network. However, it is hard to maintain the consistency of the configuration of the whole network because it needs to manage a huge amount of network information and deal with user requests that occur asynchronously. In this paper, we propose a database oriented management for asynchronous reconfiguration to achieve the consistency of configuration in SDN. We design a structure of the database to store network information and two functional components. Finally, we adopt our management system to an OpenFlow-based network, and validate that our system can manage and control an OpenFlow network via the database.

I. INTRODUCTION

As one of the new approaches to make network configuration easier and to realize a more flexible network environment, Software-Defined Networking (SDN) has emerged [1]. In SDN, network devices such as routers and switches are specialized for packet forwarding, while the control function is separated from the forwarding plane. The control function is implemented in a SDN controller that manages all network devices placed on the forwarding plane through a software program. Thus, the controller retrieves the network information from all network devices, and processes the information appropriately to decide packet-forwarding rules. If the network size becomes large, the controller has to handle a huge amount of the information, which is a network condition or control message [2]. It is hard for the controller to continue to reconfigure the whole network while maintaining the consistency of the configuration. In such cases, important problems we should be addressed are (1) to maintain the consistency against a huge amount of information updates, and (2) to operate the information update frequently and asynchronously.

In this paper, we propose a database oriented management for asynchronous and consistent reconfiguration in SDN and we also design two functional components, configuration engine and conversion driver. For our motivation, we design structures of two databases, Logical DB and Physical DB. Logical DB stores abstracted network configuration that is described by network administrator. Physical DB manages the underlying physical network information that is used for creating packet-forwarding rules for SDN switches. We also design two functional components, configuration engine and conversion driver. We address the problems mentioned above by stating that (1) a controller stores all of the information, such as the physical topology, the global network status, and the configurations into a relational database, and (2) each functional component can operate independently and asynchronously to prevent conflicts caused by the information updates. A key idea of the system is to manage up-to-date information about network configuration or the network status in databases. The database has the advantage of data processing and addresses asynchronous updates by the transaction processing. Each functional component fetches the information stored in databases and creates information about network configuration. Each component operates independently and asynchronously through databases, since the path computation function is separated from the control function. For this purpose, we also consider table structures of the databases that store the network information, such as the physical topology and packet forwarding rules. To reveal advantages of our approach, we apply the concept of database oriented management on an OpenFlow-based experimental network and validate that our system can configure an OpenFlow-based network through the databases.

The paper is organized as follows. A survey of related works is found in Section II. The detailed description of our proposed system is shown in Section III. We implement the databases and the functional components on an OpenFlow-based network in Section IV. Finally, we conclude this paper in Section V.

II. RELATED WORK

In SDN architecture, scalability of SDN controllers and consistency of configurations in the whole network become notable concerns [2][3]. The authors in [4] modified a NOX controller, which is one of the SDN controllers, to optimize multi-thread processors and investigate controller performance in four public SDN controllers including the modified NOX. In [5], the authors proposed an extensible SDN control system, the throughput of which scales with the number of CPU cores
on a single controller. They also claimed that a centralized control on the control plane may cause a bottleneck on SDN controllers. As the network size becomes large, a controller has to manage a huge number of forwarding elements and flows. As a result, configuration updates occur frequently, and then it is hard to maintain the consistency of the whole network [6]. In [7], the authors proposed a real-time verification system for the configuration of the whole network to prevent configuration failure or misses by checking forwarding rules between SDN applications and SDN controllers. Another approach to achieve a more scalable control system is a physically-distributed control plane [8]. In such a distributed control system, the control plane consists of multiple controllers that manage different domains of the whole network. To ensure the global connectivity between arbitrary two end nodes, each controller has to communicate with the adjacent controllers to exchange local configurations while maintaining the consistency. Furthermore, when updating from an old policy to a new policy across multiple switches, where two policies coexist in the whole network at the same time, then the consistency of the configurations among the switches is hard to maintain. Reitblatt et al. addressed one of the important problems in configuring multiple SDN switches [9]. To ensure the consistency of a per-packet or per-flow level, they introduced the configuration version number and timeout value.

III. A DATABASE ORIENTED MANAGEMENT FOR ASYNCHRONOUS RECONFIGURATION

In this section, we describe the overview of our database oriented management system for SDN and show the design of the databases, configuration engine, and conversion driver.

A. Overview

Figure 1 shows the overview of our database oriented management system in SDN. Our system has two databases, Logical DB and Physical DB and two functional components, Configuration engine and Conversion driver. To manage a network, a network administrator specifies abstracted network configurations, and stores the configurations into the Logical DB. Based on the network configuration stored in the Logical DB, the configuration engine determines the communication paths according to a specified routing protocol and allocates network resources to configure the physical network. After that, each engine creates low-level network information and stores them into the Physical DB. Even if two or more engines are specified in one network by the administrator, these engines can access the databases independently and asynchronously. A conversion driver translates the low-level network information into a configuration file adapting to the notation of a certain SDN controller. Finally, a SDN controller updates the flow table of each switch through a software program translated by the conversion driver.

B. Databases

1) Logical DB: Figure 2 shows the table structure of the Logical DB, which manages high-level network information described by network administrators. The high-level network information denotes an abstracted network configuration. For instance, the administrator indicates which node participates in the network, which routing mechanism is adopted, and how many network resources are assigned to the communication paths. Because of this abstracted configuration, the administrator does not have to know the global information of the physical topology and low-level configuration commands to manage SDN switches. A managed network is a network where a network administrator manages it as an administrative domain. For instance, a virtualized network operating on the physical network is a managed network. The managed network table contains the managed network name to identify managed networks in the control plane. A network consists of either nodes, networks or both. Thus, a network can contain another network. A node or a network is defined as a network element stored in the network element table. The information of node is managed in the node table. The information of network is managed in the network group table as a set of the network elements. The configuration engine stores the information of network and configuration engine. The deleted and updated fields in each table are used for checking whether the entry has been changed or not.

2) Physical DB: Figure 3 shows the table structure of the Physical DB. The information stored in the tables of this database can be categorized into either the physical information or the intermediate code. The physical information, such as the node address, is stored in the physical information tables, which are enclosed within the dashed line. The low-level control information generated by the configuration engine,
The switching actions to be applied to the flows identified by both the path ID and the link ID. The rule table manages the information of physical interfaces equipped on the node. A physical link is connected to two physical interfaces on distinct nodes. The interface list table manages the physical interfaces ID and the link ID. The physical information described above has to prepare a configuration file that is appropriate for each SDN controller, even if the network configuration is the same. The network administrator can configure the network without considering the details of the SDN controllers because the conversion driver converts the information stored in the intermediate code tables into a suitable configuration program that the SDN controller can read directly. Although various SDN controllers are proposed thus far, there is no compatibility among the SDN controllers. Thus, the network administrator has to prepare a configuration file that is appropriate for each controller, even if the network configuration is the same. The administrator can configure the network without considering the details of the SDN controllers because the conversion driver absorbs the difference among the SDN controllers. The administrator can cope with a change of the SDN controller by shifting to an appropriate conversion driver.

C. Configuration engine

The configuration engine fetches the abstracted network information from the Logical DB and processes them to create the intermediate code. Basically, each configuration engine operates independently and has just one role. Typical roles are routing calculation and resource management etc. Network administrators can decide which engines are applied to a managed network by specifying them in the high-level network information. In our management system, various network designs can be achieved by a combination of configuration engines. Moreover, it is easy to add new configuration engine to our system, and asynchronous transactions can be processed sufficiently because of the independency of the configuration engines.

D. Conversion driver

The conversion driver converts the information stored in the intermediate code tables into a suitable configuration program that the SDN controller can read directly. Although various SDN controllers are proposed thus far, there is no compatibility among the SDN controllers. Thus, the network administrator has to prepare a configuration file that is appropriate for each controller, even if the network configuration is the same. The administrator can configure the network without considering the details of the SDN controllers because the conversion driver absorbs the difference among the SDN controllers. The administrator can cope with a change of the SDN controller by shifting to an appropriate conversion driver.

IV. IMPLEMENTATION OF DATABASES AND FUNCTIONAL COMPONENTS

In this section, we implement and validate our database oriented management system on an OpenFlow network.

A. Implementation environment

Figure 4 shows the detailed settings of our experimental network. The control plane consists of the databases and functional components that are operated on Debian Squeeze.
with Linux 2.6.32. The administrator’s terminal is operated on Mac OS X 10.6.8. Two OpenFlow Switches are placed on the forwarding plane and each of these is connected directly to a client’s PC. The numbers denoted beside the switches mean port numbers of the physical interface connecting to the cable. The SDN controller, where Trema [10] is installed, is connected to a management port of each OpenFlow Switch. We develop a Logical DB and Physical DB on a MySQL Server version 5.1.66. The engine and the driver are implemented in C++.

B. Implementation of our management system

Figure 5(a) shows a sequence diagram from the time when the network administrator stores abstract network information into the Logical DB to the time when the engine stores the intermediate code into the Physical DB. Figure 5(b) shows a sequence diagram from the time when the conversion driver detects an update of the Physical DB to the time when the OpenFlow controller configures the OpenFlow switches. The procedures of managing a network in our system are described as follows.

At Step 1, the network administrator describes abstract network configurations for creating a network. To confirm that a new network is successfully created, the administrator obtains the assigned managed network ID from Logical DB. After that, the administrator has to specify nodes or networks that participate in the new network and the configuration engines to be applied to the network. At Step 2, the configuration engine continuously monitors the network information in the Logical DB. If the network group table or the engine list table has been updated, the engine fetches the network information from tables in the Logical DB related to the engine. At Step 3, the configuration engine requests physical information to the physical DB for performing path calculation or resource assignment. At Step 4, each engine creates the intermediate code as fundamental information of the configuration file for the SDN controllers, and updates the code to the intermediate code tables in the Physical DB. Table I shows snapshot of the intermediate code when we configured a network denoted in Fig.4. At Step 5, if the tables containing the intermediate code have updated, the conversion driver sends a SQL query to fetch information of the tables. The driver converts the intermediate code like Table I into the flow table. At Step 6, the conversion driver converts the intermediate code into a configuration file and transfers it to the SDN controller. Finally, Trema applies the configuration file and sends configuration commands to OpenFlow switches according to the configuration file.

C. Experimental Validation

We check whether squeeze1 and squeeze2 can communicate with each other in a new network created by configuring only the Logical DB. The details of the configuration of our experiment are shown in the previous section. After updating the Logical DB, we confirmed that two clients can reach each other by a ping command.

V. CONCLUSION

We have proposed a database oriented management for asynchronous reconfiguration in SDN. For this, we have designed structures of the databases, and two functional components. Based on our design, we have also adopted our system to an OpenFlow network, and also validated that the OpenFlow network can be managed via the databases. For future topics, we need to evaluate scalability of our system and take time to reflect on the rules.

ACKNOWLEDGMENT

This work is supported by US NSF grants CNS-1029562 and CNS-1029546, the Office of the Naval Research’s (ONR) Young Investigator Program (YIP), and a Japan NICT International Collaborative Research Grant.

### Table I. Rule table which is one of the intermediate code tables

<table>
<thead>
<tr>
<th>Rule ID</th>
<th>Node ID</th>
<th>Path ID</th>
<th>dl type</th>
<th>source ip address</th>
<th>destination ip address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>ip</td>
<td>192.168.1.2</td>
<td>192.168.1.1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>ip</td>
<td>192.168.1.2</td>
<td>192.168.1.1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>arg</td>
<td>192.168.1.2</td>
<td>192.168.1.1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>ip</td>
<td>192.168.1.2</td>
<td>192.168.1.1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>arg</td>
<td>192.168.1.1</td>
<td>192.168.1.2</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>ip</td>
<td>192.168.1.1</td>
<td>192.168.1.2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>arg</td>
<td>192.168.1.1</td>
<td>192.168.1.2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
<td>ip</td>
<td>192.168.1.1</td>
<td>192.168.1.2</td>
</tr>
</tbody>
</table>
REFERENCES


