An Elastic Distributed SDN Controller

[222 Course Project Checkpoint One]

Guo Li
A53071472
gul027@eng.ucsd.edu

Xinyu Zhang
A53095838
xiz368@eng.ucsd.edu

Liqiong Yang
A53076313
liy007@eng.ucsd.edu

1. INTRODUCTION

Software Defined Networking (SDN) provides a centralized control panel, which allows the network to be programmed by the application and controlled from one central entity, also brings the issue of reliability and scalability. Recent re- searchers have explored architectures for building distributed SDN controller, but all have an implication that the map- ping between a switch and a controller is statically configured. With this design, it is difficult for the control plane to adapt to traffic load variation. There could be the case where a controller may become overloaded, if the switches mapped to this controller suddenly received a large number of traffic, while other controllers remain idle.

There should be a migration logic that, when load imbalance occurred, it can migrate a switch from a heavily-loaded controller to a lightly-loaded one. There also should be a monitoring logic that can decide when to trigger this migration. OpenFlow currently does not support such a migrate operation and this is what we want to implement in our project.

2. WHAT WE'VE DONE SO FAR

So far in general, we have finished the following tasks:

1. Define Network Topology
2. Define Controller Migration Protocol
3. Implement a framework

3. WHAT TO DO NEXT

1. Design migration algorithm
2. Finish all the implementation task
3. Detailed evaluation and analyses

4. NETWORK TOPOLOGY

We define the network topology as the following:

1. Eight switches with a tree structure

2. Two controllers, one of which takes charge of 3 switches (No. 1, 2, 3); the other takes charge of the rest 5 switches (No. 4, 5, 6, 7, 8)

3. For every switch, we have one or several hosts.

5. MIGRATION PROTOCOL

For every controller in OpenFlow, it has three different modes: master, equal or slave. In master or equal mode, controller will receive all the asynchronous messages from switch (like Packet-In) and can modify the state of switch; in slave mode, controller only can read the status of the switches.

One switch can connect to different controllers, but only one of them should be master. So the migration of controller is actually the process of changing mode of controller: suppose controller A and B are connected to switch S, A is master and B is slave; at the end of the migration, A and B are still connected to switch S, but now A is slave and B is master.

The core logic of the protocol is very simple:

1. Upgrade B from slave mode to equal mode, which then allows it to take care of all the following flows
2. Let A finish all the previous buffered requests
3. B come up the request to become master, which will automatically transfer A to slave

However, since messages from switch are arrived at controllers out of order, so two controller must agree on a single point where the migration shall happen, so we have the following protocol.

Formally, there are four phases:

1. Change controller B from slave mode to equal mode
2. Insert and remove a dummy flow
3. A finishes all the pending request

4. B request to become master

There are more details about the four phases, which will be presented in the final report.

6. FRAMEWORK

For implementation, we have already implemented a framework based on OpenFlow, Mininet and RYU; this framework enables a basic controller scheduling logic.

7. WHAT TO PRESENT IN POSTER SESSION

Basically we will show the following parts in the poster session: problem specification, related work, design and implementation overview, evaluation results.