Implementation of a Distributed Control Plane for Ryu

1st Alba Jano
Electrical and Computer Engineering
Technical University of Munich
alba.jano@tum.de

2nd Roland Ender
Electrical and Computer Engineering
Technical University of Munich
ga59qay@mytum.de

3rd Ananya Bose
Electrical and Computer Engineering
Technical University of Munich
ananya.bose@tum.de

***Abstract*—A distributed control plane for Ryu has been implemented. Deploying a single controller in a network makes the network prone to a single point of failure. Also the network does not scale to large topologies with a single controller. A multi-controller topology has been implemented using two controllers, wherein at any point of time each switch in the network has a master controller and a slave controller as a backup. The two controllers are synchronized via TCP communication wherein, each controller shares the current bandwidth available on each link with the other. Apart from this, the controllers are also synchronized with respect to the packet_in message rate at each controller. Reliability of the system has been taken into account so that if the master controller fails, the slave controller can take over. Load balancing has been implemented in the control plane. Also, bandwidth reservation has been done per flow in the data plane.**

***Index Terms*—Software Defined Networking, multiple controllers, load balancing. reliability, bandwidth reservation**

I. Introduction

Software Defined Networking (SDN) is a novel networking paradigm that allows management of the networks in a flexible way. Most of SDN control plane designs use only one controller for the network. However, the centralized control faces problems due to increasing network demands and large scale networks. Moreover, a single controller represents a single point of failure in the network, which causes a disconnection of the switches from the controller and failure of the network to offer the required services [1].

To overcome these issues a distributed control plane, composed of multiple controllers, is proposed. Scalability, consistency, reliability and load balancing are the issues faced by the multi-controller control plane. Scalability consists in replacing the single controller with multiple controllers which can have master, slave or equal role. This implementation provides backup controllers for each running controller and resolves the single point of failure for controllers. It improves the performance of SDN networks as the switches can route the newly arrived packets at any time, even if one of the controllers fails [2].

Multi-controller design may divide the network into several domains depending on the role of the controllers. Controllers need to interact with each other to have a consistent view of the network and transmit the packets in a correct way. The interaction is done using synchronization channels among the controllers through which, the controllers should exchange their state and the control strategy to avoid conflicts for switch management. Reliability is assured if at least one controller is running. The remain running controller can change its role to the master controller of network switches and continue offering the service using the network.

Variation of the network traffic in the case when the network is partitioned into several SDN domains can produce over-loaded and under-loaded controllers in the network. The imbalanced distribution of the load among the controllers can degrade the network performance by causing high packet loss. To avoid this problem a load balancing algorithm is implemented for the controllers, which need to exchange the roles in an appropriate way [2].

In Section II we present different roles of the controllers and how these roles need to be assigned in case of multi-controller topology. The scenario with the topology specifications and features to be implemented are presented in section III. In Section IV are defined the algorithms and the way how the implementation of the features is done. Finally, results and conclusions for the defined scenario can be found in Section V.

II. Background

Deploying a single controller for an entire network is a security risk, as the controller is a single point of failure and does not scale to large networks. To avoid this, one can rather deploy more controllers. A switch may establish a connection with only one controller or with multiple controllers. Multiple controllers improve the reliability because the switch continues to operate if one of the controllers or the controller connection fails. The management of the switches is coordinated among the controllers, via out-of-bound channels. The multiple controller scenarios address fail-over and load balancing in the network. A switch must be connected to all the controllers when OpenFlow operation is initiated.

The default role of a controller is OFPCR_ROLE_EQUAL. The controller is equal to the other controllers that have the same state and have full access to the switch. Asynchronous messages, as packet_in, are received by the controller and it can send controller-to-switch commands to modify the state of the switch.

The controller can be in OFPCR_ROLE_SLAVE role, when it can only read from the switch and does not receive the asynchronous messages, apart from port status messages. The controller-to-switch commands are denied for this controller. If the controller sends controller-to-switch messages an OFPT_ERROR message will be received a reply from the switch and the switch state will not be modified. Other controller-to-switch messages, such as OFPT_ROLE_REQUEST are processed normally.

When the controller is in the role of OFPCR_ROLE_MASTER it has full access to the switch. This role is similar with OFPCR_ROLE_EQUAL, however, there should be only one controller with the role of OFPCR_ROLE_MASTER per switch and there can be multiple controllers with the role of OFPCR_ROLE_EQUAL per switch.

The controller may change its role by sending an OFPT_ROLE_REQUEST message to communicate with the switch, which should remember the role of the defined controller. A generation_id is generated each time the controller changes its role. The role of the controller cannot be changed by the switch directly, but there should be a request from the controller to change the role. The request for changing controller roles need to be coordinated among the controllers, to avoid conflicts related to the OFPCR_ROLE_MASTER per switch. When the switch has already a controller in Master role then the other controllers can be only in the Slave role, and not in Equal or Master role [3].

III. Scenario and Project Proposal

A multi-controller implementation has been done with two controllers. At the start, controller C1 is connected to switch S1 as master and S2 as slave. Controller C2 is connected to S2 as master and S1 as slave. S1 is connected to two hosts h1 and h2. S2 is connected to two hosts h3 and h4. There is a TCP communication between the two controllers, wherein each controller sends to the other the information of current bandwidth available on the links: h1 to S1, h2 to S1, h3 to S2, h4 to S2, S1 to S2. Load balancing [4] has been done in the control plane, wherein if the number of packet_in messages per five seconds at one controller becomes higher than that at the second controller by a specific threshold, the second controller will become the master of the switch that had the first controller as master. When the difference in the packet_in message count per five seconds again falls below below the threshold, the previous configuration is restored. Controller reliability has also been implemented, wherein, if one controller fails, the other controller will become the master of both the switches. Bandwidth reservation per flow has been implemented in the data plane. The bandwidth reserved on each link between a host and a switch and on the link between the 2 switches have been taken as some specific values. Flows will go through as long as the bandwidth constraint is satisfied.

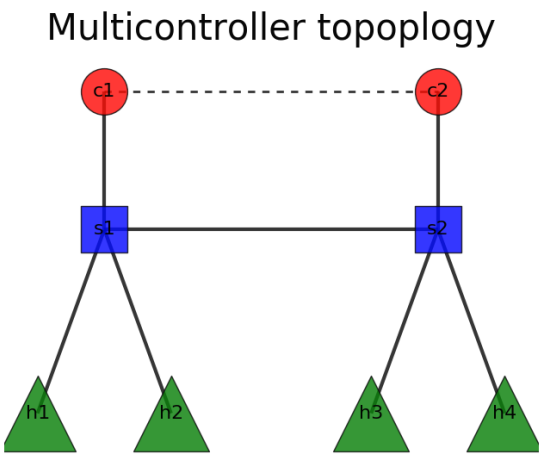

Fig. 1. Multi-controller topology

IV. Algorithm and Implementation

In order to avoid any inconsistencies between the different controllers (e.g., different decisions which would lead to a routing loop) one must ensure that the controllers are properly synchronizing their states. The goal of this project is to implement such a synchronization strategy for Ryu. To realize the synchronization among controllers a TCP connection is used, where sockets are open in both controllers. One of the controllers has the role of the server which listen and send messaged and the other controller has the role of the client. Implementation of the TCP connections are made inside the controller class and run in parallel with the process of controlling the switches by the controller, via threading which is implemented in the monitor function. TCP connection is establish when both the controller start running in different ports using ryu-manager. Controllers send periodically messages towards the other controller, meaning that they are running and a failure has not happened in the network. However, when one of the controllers does not transmit data towards the other controller, it means that a failure has happened in the network and the remaining running controller must become the master of all the switches in the network, to assure correct routing of the packets. A flag self.tcp_connection_on indicates the state of the other controller, that becomes false when the other controller fails, and is used inside the monitor function to change the role of the remaining controller for the switches.

Initially the role of controllers is set when an ryu.controller.dpset.EventDP event notifies the connection of the switches, using the function on_dp_change, which sets the role of the controller for each switch connected. The role of the controller is set using and OFPRoleRequest(datapath, role=None, generation_id=None) which is generated from the controller, defining its role for each switch of the network. on_role_reply function is used to get the role reply messages from the switch towards the controller to assure that the controller has the role that he requested.

In case of failure of one controller scenario, the remaining controller sends role request to be the master of the switches of the network and therefore, the available controller gets the packet-in from all the switches, and can modify their states. The datapath of available switches of the network is stored in a list at switch_features_handler function, so each controller knows the switches of the network.

Moreover, to handle the load balancing problem among the controllers a change of the controller role is done each time a controller is overloaded compared to the other controller. To measure the load of the controller the number of packet_in for a specific time interval is calculated for each controller and compared among each other using a specific threshold. In case there is a overloaded controller then the other controller need to to change the role as master of all the switches. Special attention needs to paid in this case, because the controllers need to be synchronized while changing the role of the switches. First the overloaded controller needs to change the role as slave for all the switches, to avoid conflicts that may happen due to request to have two master controllers for the same switch. After the load is balanced and the network is not facing issues such as high packet loss, the roles of the controllers need to go to the initial state, and each controller need to be the master of its own SDN domain. The implementation of the controller role switching is done using an self.role list which stores the current roles of the controllers.

For the purpose of bandwidth reservation in the data plane, as mentionned in the section Scenario and Project Proposal, the bandwidth reserved on each link between a host and a switch and on the link between the 2 switches have been taken as some specific values. The current bandwidth available on each link is obtained from the TCP communication at each controller. When a new flow comes in, the first packet creates a packet_in message at the master controller. Inside the packet_in handler it is first checked if the source host of the flow is in the SDN domain of the controller i.e. if the controller is the master of the switch the source host is connected to. If yes, it is checked if on each link between the source and destination, sufficient bandwidth is available. If yes, the bandwidth on each link on the path is reduced by the bandwidth required by the flow. In this case the flow is added and goes through and create a packet_in at the second controller, considering S1 has C1 as master and S2 has C2 as master. There will be no bandwidth reduction at this controller since the source host is not in its domain. In this way it is ensured that the bandwidth is not reduced twice for a single flow by the two controllers. If sufficient bandwidth is not available on any link on the path from the source to the destination, the flow is dropped.

For each of the two controllers, in the packet_in handler, a packet_in counter is incremented by one and the updated value is written to a csv file. Load-balancing has been done in the control plane with respect to the number of packet_in messages at each controller. A thread for monitoring traffic is started in the init function. Inside the thread, an infinite while loop is run. At every iteration of the loop, after a wait of 5 seconds, the current packet_in counter values for controller 1 and 2 are read from csv file. The packet_in count during the last interval (5 seconds) is obtained for both the controllers. If the absolute value of the difference in the packet _in count during the last interval between the two controllers is greater than a specified threshold, then if the last interval packet_in count for controller 1 is greater than the last interval packet_in count for controller 2, then controller 2 will become the master of switch S1; if the last interval packet_in count for controller 2 is greater than the last interval packet_in count for controller 1, then controller 1 will become the master of switch S2. If the absolute value of the difference in the packet _in count during the last interval between the two controllers is within the specified threshold, then if C1 was made the master of S2 or C2 was made the master of S1 previously, then C1 will go back to being the master of S1 and C2 will go back to being the master of S2.

Algorithm 1 Load balancing algorithm

```
1:  while true do
2:    sleep for 5 seconds
3:    packet_in_counter_c1_curr, packet_in_counter_c2_curr
      extracted from the csv file
4:    packet_count_last_interval_c1 =
      packet_in_counter_c1_curr - last_packet_in_counter_c1
5:    packet_count_last_interval_c2 =
      packet_in_counter_c2_curr - last_packet_in_counter_c2
6:    if abs(packet_count_last_interval_c1-
      packet_count_last_interval_c2)>thres_packet_in_rate
      then
7:      if packet_count_last_interval_c1 >
        packet_count_last_interval_c2 then
8:        C2 will become the master of S1
9:      else if packet_count_last_interval_c2 >
        packet_count_last_interval_c1 then
10:       C1 will become the master of S2
11:     end if
12:   else if abs(packet_count_last_interval_c1-
      packet_count_last_interval_c2)<thres_packet_in_rate
      then
13:     if C1 is the master of all the switches or C2 is the
        master of all the switches then
14:       C1 will become the master of S1 and C2 will
          become the master of S2
15:     end if
16:   end if
17:   last_packet_in_counter_c1=packet_in_counter_c1_curr
18:   last_packet_in_counter_c2=packet_in_counter_c2_curr
19: end while
```

V. Results and Conclusion

The above implementation of multi-controller distributed control plane solves the presented issues when using multiple controllers. Scalability of the network is assured using two

controllers, wherein each controls a domain of the SDN network. One of the controllers is the master of the domain and the other one slave, to provide a backup in case of failure of the master controller.

The single point of failure is avoided by synchronizing the controllers through TCP communication and switching the roles from slave to master in case of a failure of the controller defined by the absence of hello messages at TCP during TCP communication. This is noticed when we kill one of the controller and let the other one running. Previously the controllers were getting packet_in just from one switch and they were not aware of asynchronous messages sent from the other switch, for which they are at the role of slave. In the moment when one of the controllers is killed the running controller will send role request messages to be the master of both the switches and received packet_in from both the switches.

Load balancing in the control plane has been tested. When the packet_in rate at one controller exceeds that at the other controller by the threshold, the other controller takes over as master of the switch. Also when it falls below the threshold, C1 and C2 are restored as the master controllers of S1 and S2 respectively.

For the purpose of testing bandwidth reservation logic in the data plane, it has been tested that the current bandwidth available on each link in the network is synchronized between the two controllers via the TCP communication. Using iperf UDP traffic it has been observed that when the bandwidth exceeds on any link on the path between the source and the destination, the flow does not go through and gets dropped.

To conclude, a distributed control plane has been implemented for Ryu where two controllers are synchronized via TCP communication with respect to the bandwidth on each link and via a csv file with respect to the packet_in message rate. The implementation provides reliability, load balancing in the control plane and bandwidth reservation per flow in the data plane.

ACKNOWLEDGMENT

We would like to thank our supervisors Amaury Van Bemten and Nemanja Deric, Lehrstuhl fr Kommunikationsnetze for providing us the required support to complete the Laboratory successfully.

REFERENCES

[1] Nunes, Bruno Astuto A., et al. "A survey of software-defined networking: Past, present, and future of programmable networks." IEEE Communications Surveys Tutorials 16.3 (2014): 1617-1634.

[2] T. Hu, Z. Guo, P. Yi, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," in IEEE Access, vol. 6, pp. 15980-15996, 2018. doi: 10.1109/ACCESS.2018.2814738.

[3] "OpenFlow Switch Specification Version 1.3.5 (Protocol version 0x04)" https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf

[4] Zhou, Yaning, et al. "Load balancing for multiple controllers in SDN based on switches group." Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific. IEEE, 2017.