INFO8011: Network Infrastructures

Software-Defined Networking in Data Centers

Maxime Goffart
180521

Olivier Joris
182113

Academic year 2021 - 2022

1 General overview

1.1 Spanning Tree Controller

This controller, as requested, builds a spanning tree in order to handle cycles in the network. To implement this solution, we first needed to discover the topology of the network. To do so, we used the functions `get_host`, `get_link`, and **get_switch** of Ryu.
Then, based on the discovered topology, we built a graph of the topology where the vertices are the switches and the edges are the links between the switches. Afterwards, to compute the minimal spanning tree, we used Prim's algorithm.

When packets are received at the controller, through `switch_in_handler` method, we check the source and the destination of each packet.
If the source is one of the host and the destination is broadcast, we broadcast the packet to the switches connected to the one that received it in the minimal spanning tree. Thus, the switches to which we sent the packet will send the packet to the controller that will, once again, do the same processing. At the same time, we set flows in the different switches so the controller will be contacted only once for a particular type of packets and it will reduce the RTT for similar requests.
If the source and the destination are both hosts, we compute a path between them using a depth-first search algorithm. Of course, the path is limited to links in the minimal spanning tree. When the path is computed, we set flows in the switches along the path so the controller will be contacted only once for a particular type of packets and it will reduce the RTT for similar request.

2 Tests

2.1 Spanning Tree Controller

If we ping 10 times each host from `h0`, we obtained the results of table 1. The simple commands used are available in the file `test_spanning_tree.sh`.

Source-Dest	**Mean**	**MDEV**	**1st ping**
H0-H1	294ms	236ms	1003ms
H0-H2	485ms	201ms	1091ms
H0-H3	651ms	497ms	2024ms
H0-H4	721ms	257ms	1629ms
H0-H5	734ms	298ms	1629ms
H0-H6	718ms	255ms	1484ms
H0-H7	713ms	237ms	1425ms
H0-H8	720ms	219ms	1380ms
H0-H9	719ms	248ms	1463ms
H0-H10	720ms	249ms	1467ms
H0-H11	723ms	254ms	1485ms
H0-H12	717ms	251ms	1468ms
H0-H13	702ms	211ms	1337ms
H0-H14	716ms	243ms	1447ms
H0-H15	712ms	248ms	1456ms

Table 1: Results of 10 pings between each host and `h0`

We can see that the first ping always take a lot of time. This is explained because the first ping we will have as consequence to send multiple packets to the controller. First, it will send packets to the controller related to the ARP request to learn the MAC address of the destination. Then, it will send packets to the controller related to the icmp messages which will require the computation of paths between the host and the destination. All the processing in the controller is done in Python which is well known to be a slow language. Furthermore, these tests were performed on Maxime's computer which had the vm lock with an execution cap at 10% (see in section 3 the explanation).
Yet, the other pings have a RTT around the mean value. The high values of MDEV are explained by the fact that the first ping take a lot of time thus it increases the values of MDEV.
Another reason for the high value is the fact that each link has a delay of 50ms as set in the Mininet topology provided. Finally, we should keep in mind that we are running in a virtual environment and not on physical devices thus the measurements are not the most precise.

3 Feedback

3.1 Main difficulties

We encountered multiple difficulties when working on this assignment. Here is a list of the difficulties we encountered:

- Instabilities of the virtual machine, Mininet, and Ryu. For instance, Maxime had to cap the execution of the processor for the virtual machine at 10% or the code would not work. He lost some time trying to understand the issue.
- Most documentations, even the official book, on how to use Ryu is for OpenFlow 1.3 while we were blocked to OpenFlow 1.0. The differences between the 2 versions are minors but they are resulting in lose time that could be used to improve our understanding of SDN. Also regarding documentations, they are not always very well done.

These difficulties have the consequence that we have the impression that we spent more time on details related to Ryu and different versions of OpenFlow than really improving our understanding of SDN. Furthermore, these difficulties increased the difficulty of the project uselessly.

3.2 Time spent on the project

To be filled

3.3 Possible improvments

Here is a list of possible improvements for the project:

- Switching to OpenFlow 1.3 because, as mentioned previously, most documentations on Ryu are using OpenFlow 1.3 and we will not lose time on details related to which version of OpenFlow we are using.
- Maybe using something else than Ryu because we had issues related to it (e.g. needing to cap the execution of the processor or some functions of Ryu would return anything).