UNIVERSITY OF VIENNA

Summer Internship

Weekly report #1

Amir Sabzi
+989373107525
97.amirsabzi@gmail.com
August 2020

Letter of Submittal

To:	Prof. Stefan Schmid
From:	Amir Sabzi
Date:	August 7, 2020
Re:	Work Report: Report #1

Hi Professor,

I decide to write short weekly reports of what I've done during the week. I think this will help me to stay focused on the tasks, also it will Clarify the issues and it is a good way to record my ideas. And I think this weekly reports will provide you proper information of my progress in the tasks that you've assigned to me.

Sincerely

Amir Sabzi

Table of Contents

List of Figures

1 Implementation of DPID Covert Channel

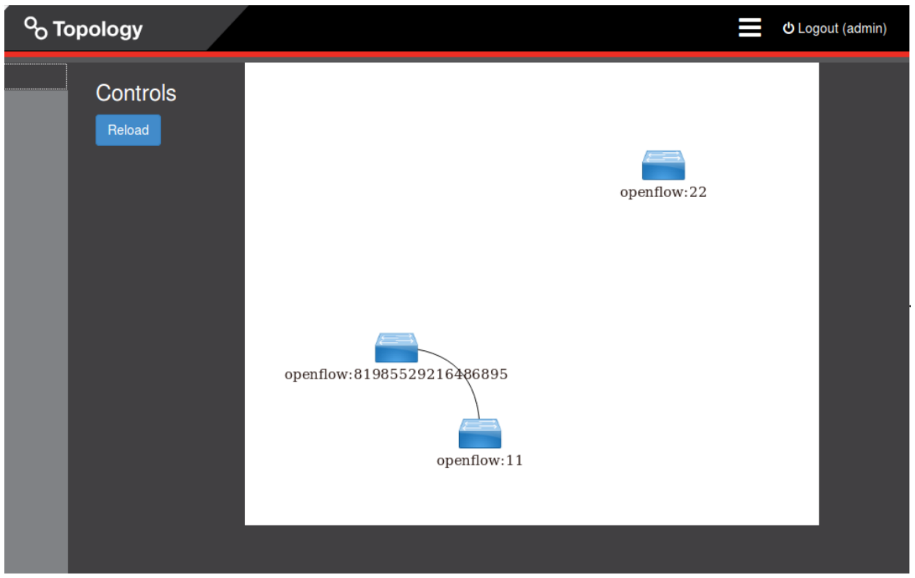
For this part I run two virtual machines, such that one of them contains the switches and another run OpenDaylight controller. I've used dlux feature of OpenDaylight controller to provide a GUI, so I could monitor connectivity of switches. You mentioned in the paper that you used ONOS controller as control plane of the test network, I decided to use OpenDaylight controller, in order to study the behavior of this common controller.

Unlike the ONOS controller which denies the connection with the second switch with same DPID, ODL accepts the second connection and disconnects the first one. This uncommon reaction is easily exploitable to implement a denial of service attack. In order to show that I made a simple topology consisted of four switches, *S11*, *S1*, *S2*, *S22*. In the topology *S1* is connected to the *S1* and *S2* is connected to the *S2*. at the first, I change DPID of *S1* by the following command:

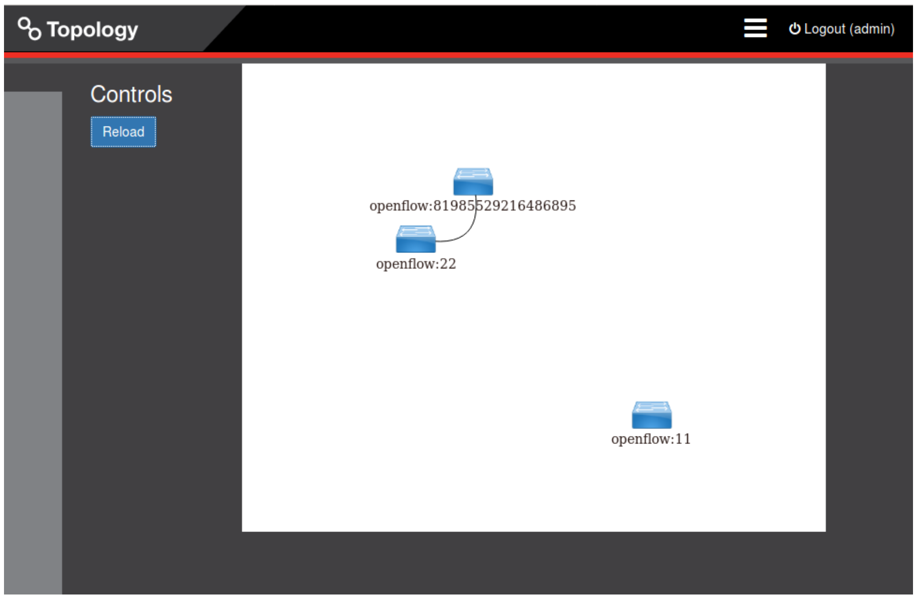
```
$ovs-vsctl set bridge S1 other-config:datapath-id=<UID>
```

Network state has demonstrated in the figure 1. After that, we connect the second switch, *S2*, to the controller with the datapath-id of *S1*. As you can see in the figure 1, the first switch was disconnected and second one was connected.

One important problem in the implementation of the covert channel is the short reconnection timeout of the OpenVirtualSwitch that make it difficult



(a) Before connection of *S2*



(b) After connection of *S2*

Figure 1: Network status during connection procedure of *S2*

to communicate properly in this short time. So we should increase this timeout. I checked several ways to do that. However after discussion with Mr. Thimmaraju in the last weed, I realized that I should modify source code. So I started to search for that and I find the related variables in the source code and then change them. Thus the Implementation is almost completed.

2 Covert Channel in programmable data planes

2.1 Feasibility of exploiting DPID in the programmable data plane

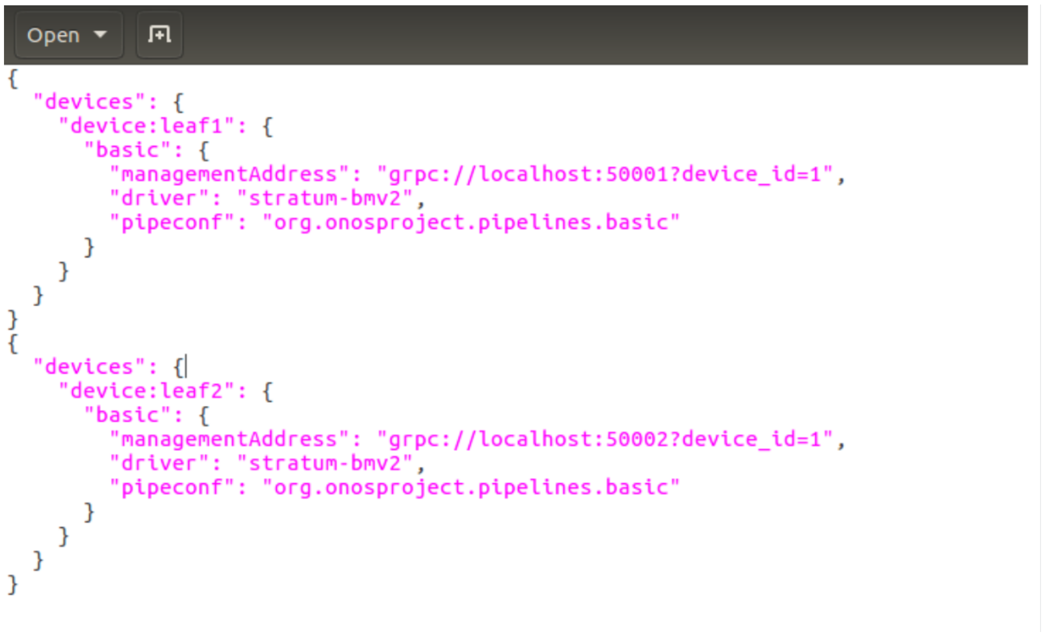
After facing the described problems in the first task, I decided to don't stop at this point. So I tried to check the feasibility of applying the same idea explained in the paper to a p4 switch controlled by the ONOS controller. I installed a VM contains the ONOS controller and p4 compiler, then I created a simple topology in mininet and tried to find out how the controller discriminates different switches.
Using the guidance[1] of Mr. Thimmaraju, I found out the handshake between switches and the controller in p4runtime is initiated by the controller, and controller assigns each switch an unique ID. And save the configuration of

[1]he provided me a bachelor thesis about this area

each switches locally, e.g. The ONOS controller will create a *netcfg.json* file for each of the switches. You can see contents of this file in the figure 2.
As it's demonstrated in the figure 2, each switch has the dedicated management Address used to discriminate between different switch in the network.
According to the p4runtime specification, the controller opens one *stream-channel* per switch, then will send *StreamMessageRequest*. This message sets the *device_id* and *role_id* for each one of the switches. Finally the controller configure each target via the *SetForwardingPipelineConfig* RPC.
Thus, because the handshake is initiated by the controller in p4runtime, **it's NOT possible to exploit switch identification to implement a covert channel in the p4-enabled switches managed by the ONOS controller.**

2.2 A possible way to implement a covert channel in the programmable data plane

In the meeting with Mr. Thimmaraju, he advised me to read one of his papers, "Outsmarting Network Security with SDN Teleportation", in order to find other possible ways to implementing a cover channel in the programmable data plane.
I read the paper and I think using the flow reconfiguration technique can be a good method. Since independent of any type of switch and controller



```
{
  "devices": {
    "device:leaf1": {
      "basic": {
        "managementAddress": "grpc://localhost:50001?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.pipelines.basic"
      }
    }
  }
}
{
  "devices": {
    "device:leaf2": {
      "basic": {
        "managementAddress": "grpc://localhost:50002?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.pipelines.basic"
      }
    }
  }
}
```

Figure 2: *netcong.json* file in the *StratumBmv2Switch*

that we use in our network, flow tables should be consistent. So I think we can use path update or path reset to implicitly communicate between the switches in the programmable data plane.

If you think it would be a good idea, I can try to implement flow reconfiguration technique in a p4-enabled switch. however, I remember you emphasized on using machine learning to create/modify the covert channel. Currently, I don't know how machine learning will work with this method. But if you think that it's needed, we can talk in the coming days.

Best regards.

Amir Sabzi