Laboratory 3 OpenFlow Protocol

Part 1: Introduction to Openflow and Software Defined Networks (SDN)

1.1 SDN and OpenFlow

Software-Defined Networking proposes a separation of the network control plane from the forwarding plane. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol was standardized to provide the communication between the SDN Controller (Control Plane) and the network devices (Data Plane) – Figure 1. OpenFlow is an open interface for remotely controlling the forwarding tables in network switches, routers, and access points. Upon this low-level primitive, researchers can build networks with new high-level properties. For example, OpenFlow enables more secure default-off networks, wireless networks with smooth handoffs, scalable data center networks, host mobility, more energy-efficient networks and new wide-area networks [*https://github.com/mininet/openflow-tutorial/wiki*].

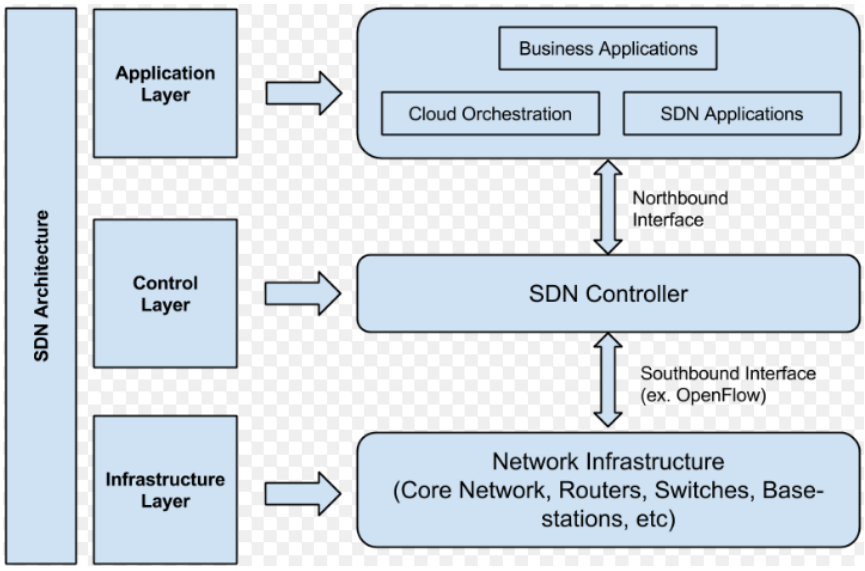

Figure 1: SDN Architecture [*http://www.openairinterface.org/?page_id=466*]

Software defined networking provides the following capabilities [*https://technet.microsoft.com/en-us/windows-server-docs/networking/sdn/software-defined-networking*]:

The ability to abstract applications and workloads from the underlying physical network, which is accomplished by virtualizing the network. For example, software defined networking provides virtual abstractions for physical network elements, such as IP addresses, switches, and load balancers.

The ability to centrally define and control policies that govern both physical and virtual networks, including traffic flow between these two network types.

1.2 OpenFlow messages.

Some examples of OpenFLow messages are presented in the next tables. In Table 1 general purpose messages are shown:

Message	Type	Description
Hello	Controller->Switch	following the TCP handshake, the controller sends its version number to the switch.
Hello	Switch->Controller	the switch replies with its supported version number.
Features Request	Controller->Switch	the controller asks to see which ports are available.
Set Config	Controller->Switch	in this case, the controller asks the switch to send flow expirations.
Features Reply	Switch->Controller	the switch replies with a list of ports, port speeds, and supported tables and actions.
Port Status	Switch->Controller	enables the switch to inform that controller of changes to port speeds or connectivity. Ignore this one, it appears to be a bug.

Table 1: General purpose OpenFlow messages

In Table 2 the messages used for flow management are shown:

Message	Type	Description
Packet-In	Switch->Controller	a packet was received and it didn't match any entry in the switch's flow table, causing the packet to be sent to the controller.
Packet-Out	Controller->Switch	controller send a packet out one or more switch ports.
Flow-Mod	Controller->Switch	instructs a switch to add a particular flow to its flow table.
Flow-Expired	Switch->Controller	a flow timed out after a period of inactivity.

Table 1: Flow management messages

Part 2. Practice

[*https://github.com/mininet/openflow-tutorial/wiki/Learn-Development-Tools*]

Step 1: Implementing Mininet network

1) Open VirtulaBox virtualization environment and start the Mininet virtual machine (Before starting the machine check the network settings for the virtual machine – **bridge adapter must be selected**). Login to the Mininet machine using username **`mininet`** and password **`mininet`**.

2) Connect from Linux terminal on the local machine to the Mininet virtual machine using SSH:

```
$ ssh -X mininet@MininetMachine'sIPaddress
```

The Mininet machine's IP address can be discovered with the `ifconfig` command in the linux terminal.

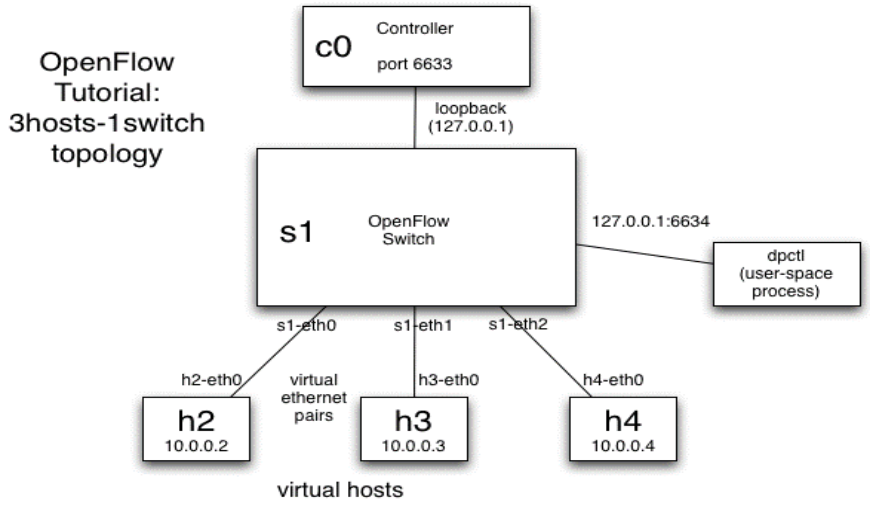

Figure 2: Mininet network topology [*https://github.com/mininet/openflow-tutorial/wiki/*]

3) Start the Mininet program with the topology given in Figure 2:

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

This tells Mininet to start up a 3-host, single-(openvSwitch-based) switch topology, set the MAC address of each host equal to its IP, and point to a remote controller which defaults to the localhost.

Q1. Analyze the network topology using the basic Mininet commands (`mininet>help` to find the commands).

Step 2: ovs-ofctl tool for manually installing flows

`ovs-ofctl` is a utility that comes with Open vSwitch and enables visibility and control over a single switch's flow table. It is especially useful for debugging, by viewing flow state and flow counters. Most OpenFlow switches can start up with a passive listening port, from which you can poll the switch, without having to add debugging code to the controller.

4) Create a second SSH window to the Mininet machine and run:

```
$ sudo ovs-ofctl show s1
```

The `show` command connects to the switch and dumps out its port state and capabilities.

To display data about the flow-table `dump-flows` command must be used.

```
$ ovs-ofctl dump-flows s1
```

The flow-table must be empty because we haven't started the controller yet.

5) Try to ping h2 from h1 in the Mininet machine:

```
mininet> h1 ping -c3 h2
```

Q2. Does ping work? Why?

As we saw before, the flow table is empty and the controller is not connected to populate the flow table, that's why the switch does not know how to deal with the incoming traffic. `Ovs-ofctl` can be used to manually insert flow entries in the switch's forwarding table. To manually install the necessary flows, type the following commands in the SSH terminal:

```
# ovs-ofctl add-flow s1 in_port=1,actions=output:2

# ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

This will forward packets coming at port 1 to port 2 and vice-verca. Verify by checking the flow-table

```
# ovs-ofctl dump-flows s1
```

Run the ping command again in mininet console:

```
mininet> h1 ping -c3 h2
```

Q3. Does ping work? Why?

Q4. Check the flow-table again and look the statistics for each flow entry.

Step 3: Running a remote controller

6) Open a new `ssh` window to Mininet Machine. To view control traffic using the OpenFlow Wireshark dissector, first open wireshark on the `ssh` terminal:

```
$ sudo wireshark &
```

In the Wireshark filter box, enter the **`of`** value for the filter, then click `Apply`. In Wireshark, click `Capture`, then `Interfaces`, then select `Start` on the `loopback interface (lo)`. For now, there should be no OpenFlow packets displayed in the main window.

7) Now, with the Wireshark dissector listening, start the OpenFlow reference controller. This starts a simple controller that acts as a learning switch:

```
$ controller ptcp:
```

Q5. Analyze the OpenFlow messages displayed in Wireshark. Expand the messages and analyze their content.

8) Now, we'll view messages generated in response to packets. Ping command will be used to trigger the flow-table update OpenFlow messages.

Before that, update wireshark filter to ignore the echo-request/reply messages (these are used to keep the connection between the switch and controller alive): Type the following in your wireshark filter, then press apply:

```
of and not (of10.echo_request.type or of10.echo_reply.type)
```

For older version of the wireshark plugin, you may need to use a slightly different syntax:

```
of && (of.type != 3) && (of.type != 2)
```

Run a ping to view the OpenFlow messages being used. You will need to run this without having any flows between h1 and h2 already installed, e.g. from the "add-flows" command above. Remove any existing flows with:

```
$ sudo ovs-ofctl del-flows s1
```

It's also recommended to clean up ARP cache on both hosts, otherwise you might not see some ARP requests/replies as the cache will be used instead:

```
mininet> h1 ip -s -s neigh flush all
mininet> h2 ip -s -s neigh flush all
```

Do the ping in the Mininet console:

```
mininet> h1 ping -c1 h2
```

Q6. Analyze the OpenFlow messages displayed in Wireshark. Analyze their content.

Use `iperf` to benchmark the controller

9) `iperf` is a command-line tool for checking speeds between two computers.

Here, you'll benchmark the reference controller; later, you'll compare this with the provided hub controller, and your flow-based switch (when you've implemented it). In the mininet console run :

```
mininet> iperf
```

This Mininet command runs an iperf TCP server on one virtual host, then runs an iperf client on a second virtual host. Once connected, they blast packets between each other and report the results.

Now compare with the user-space switch. In the mininet console:

```
mininet> exit
```

10) Start the same Mininet with the **user-space switch**:

```
$ sudo mn --topo single,3 --mac --controller remote --switch user
```

Run one more `iperf` test with the reference controller:

```
mininet> iperf
```

Q6. Which is the difference between the two scenarios?

With the user-space switch, packets must cross from user-space to kernel-space and back on every hop, rather than staying in the kernel as they go through the switch. The user-space switch is easier to modify (no kernel oops'es to deal with), but slower for simulation.

References:

- http://mininet.org/walkthrough/
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- https://github.com/mininet/mininet/wiki/Documentation
- https://openmaniak.com/iperf.php
- https://github.com/mininet/openflow-tutorial/wiki
- https://technet.microsoft.com/en-us/windows-server-docs/networking/sdn/software-defined-networking
- http://www.openairinterface.org/?page_id=466