The University of Queensland
School of Information Technology and Electrical Engineering

Semester 2, 2017

COMS4200/7200 – Practical 5

Creating ONOS Apps

Introduction

This Prac gives a hands-on introduction on how to create ONOS apps.

Exercise 1 (Creating an Empty Application)

Step 1: You have to create the ONOS bundle project. The commands used for this purpose will create a folder based on the name of your application. So select a suitable directory (such as *Applications* in the case SDN-Hub Virtual Machine) and create ONOS bundle via **one** of the following ways:

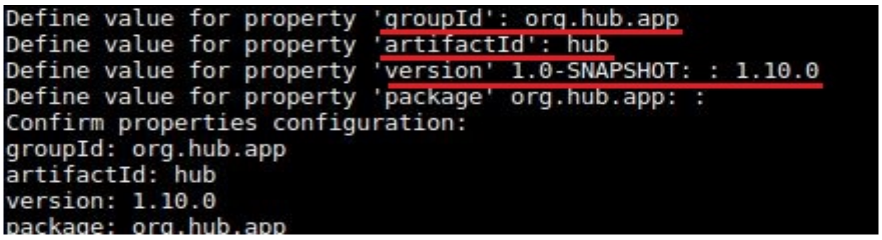
```
$ onos-create-app
```

Or alternatively

```
$ mvn archetype:generate -DarchetypeGroupId=org.onosproject
-DarchetypeArtifactId=onos-bundle-archetype
-DarchetypeVersion=<ONOS_Version>-SNAPSHOT
```

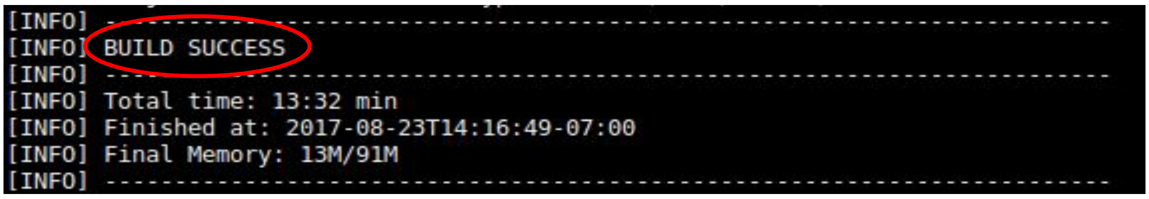
If you don't specify the version, by default it would go the newest RELEASE version of the *onos-archetypes* plugin.

Step 2: You will be prompted to define the *groupId*, *artifcatId*, *packages* …etc. You only need to specify property *groupId*, property *artifcatId* and *property* version. In this prac, we define Property groupId: ***org.hub.app*** , Property artifactId: **hub** and Property version: **1.10.0** as shown below:



```
Define value for property 'groupId': org.hub.app
Define value for property 'artifactId': hub
Define value for property 'version' 1.0-SNAPSHOT: : 1.10.0
Define value for property 'package' org.hub.app: :
Confirm properties configuration:
groupId: org.hub.app
artifactId: hub
version: 1.10.0
package: org.hub.app
```

Leave the rest empty and continue hitting enter until you see the project is created successfully as shown below:



```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13:32 min
[INFO] Finished at: 2017-08-23T14:16:49-07:00
[INFO] Final Memory: 13M/91M
[INFO] ------------------------------------------------------------------------
```

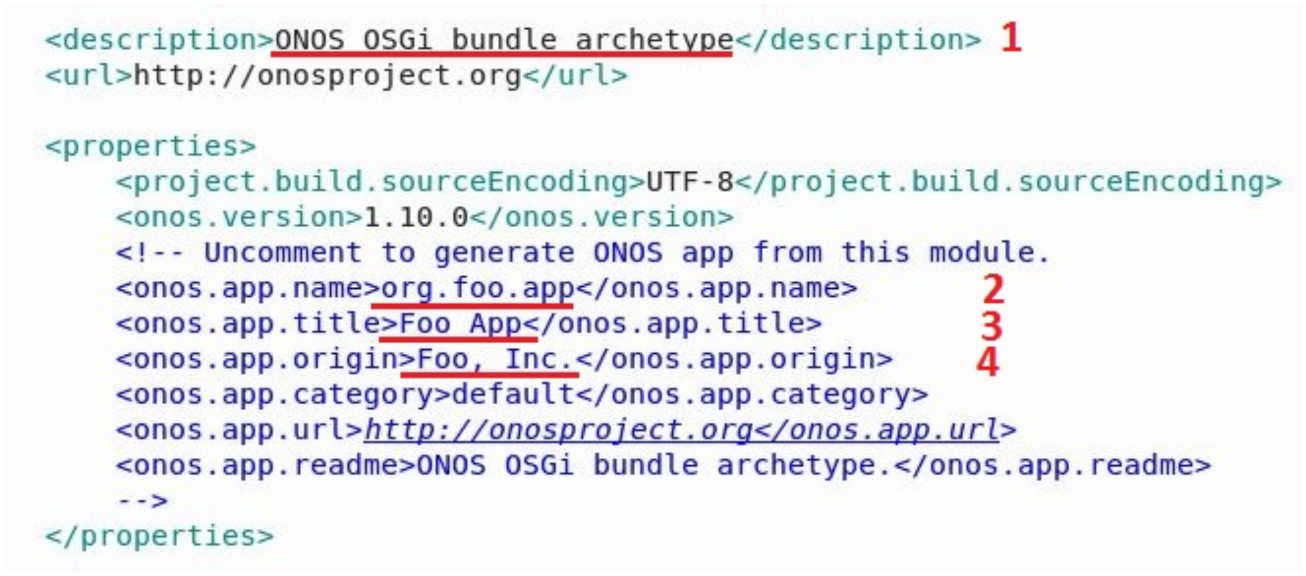
Step 3: In the next step, we need to edit the project's *pom.xml* file. A **Project Object Model** or **POM** is the fundamental unit of work in Maven. It is an XML **file** that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. The *pom.xml* file is located in the folder where your application is created in (the folder name is similar to what you gave to your application), go to the folder via the following command:

```
$ cd hub
```

Then open the *pom.xml* in a text editor such as *gedit* via the following command:

```
$ -
```

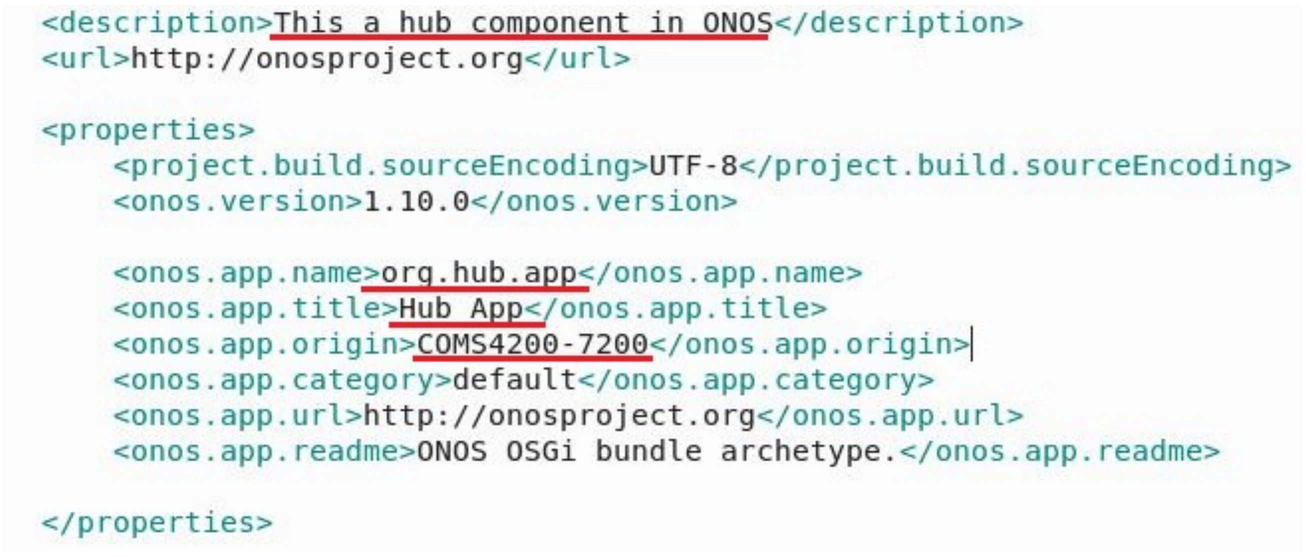
Edit the following lines as highlighted and replace the values as per the image shown on the next page:



```
<description>ONOS OSGi bundle archetype</description> 1
<url>http://onosproject.org</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <onos.version>1.10.0</onos.version>
    <!-- Uncomment to generate ONOS app from this module.
    <onos.app.name>org.foo.app</onos.app.name>         2
    <onos.app.title>Foo App</onos.app.title>           3
    <onos.app.origin>Foo, Inc.</onos.app.origin>      4
    <onos.app.category>default</onos.app.category>
    <onos.app.url>http://onosproject.org</onos.app.url>
    <onos.app.readme>ONOS OSGi bundle archetype.</onos.app.readme>
    -->
</properties>
```

Then uncomment the block as mentioned in the file and change as following:



```
<description>This a hub component in ONOS</description>
<url>http://onosproject.org</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <onos.version>1.10.0</onos.version>

    <onos.app.name>org.hub.app</onos.app.name>
    <onos.app.title>Hub App</onos.app.title>
    <onos.app.origin>COMS4200-7200</onos.app.origin>
    <onos.app.category>default</onos.app.category>
    <onos.app.url>http://onosproject.org</onos.app.url>
    <onos.app.readme>ONOS OSGi bundle archetype.</onos.app.readme>

</properties>
```

Then save the file and exit.

Step 4: Now, build the application using maven via the following command and wait until you see (“*BUILD SUCCESS*”) (**Note**: You need to be inside the `/hub` folder to run the below command)

`$ mvn clean install` (or by using its shortcut: `$mci`)

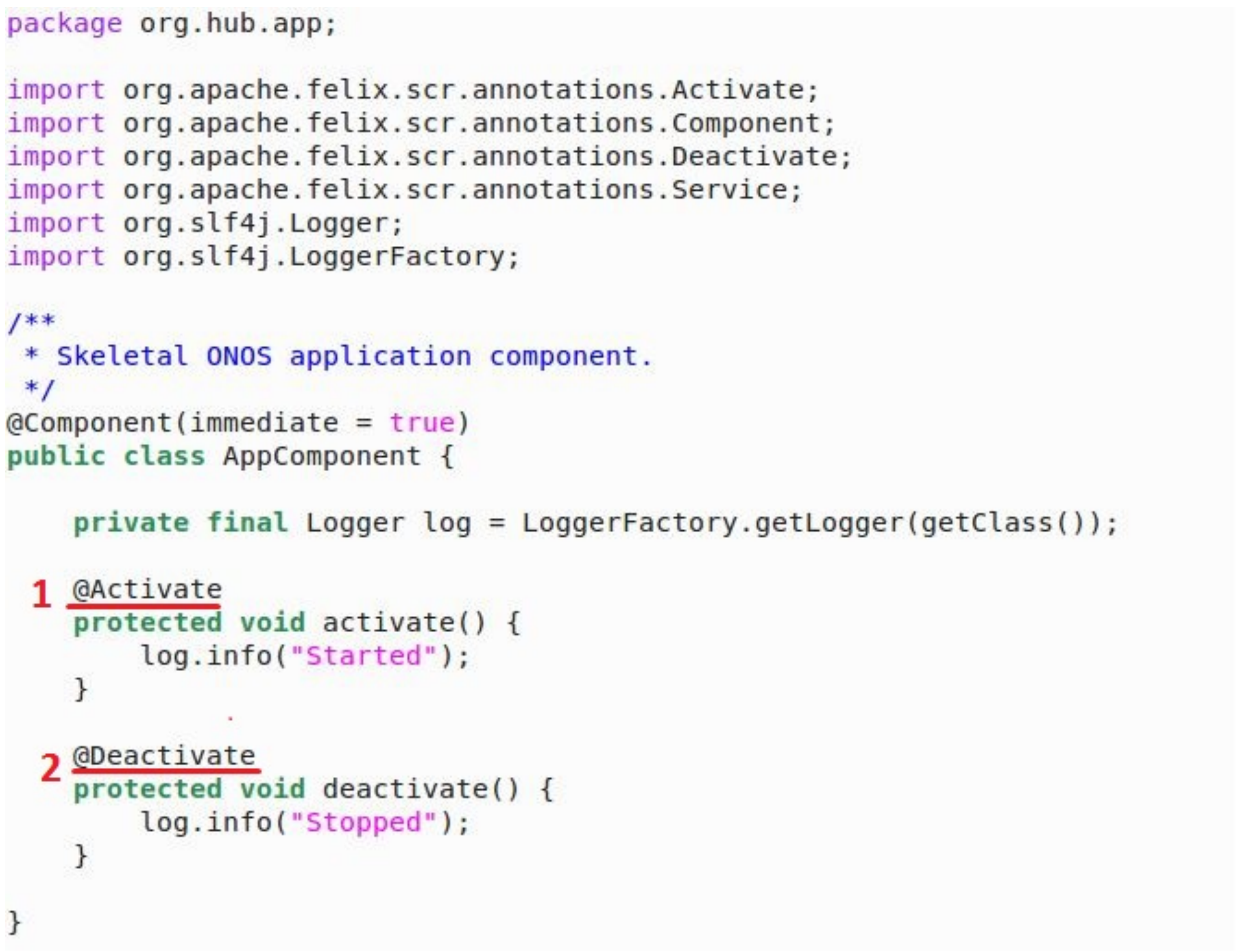
Step 5: If there is no error, navigate to the java source file:

```
$ cd src/main/java/org/hub/app
```

Step 6: In this folder you will see (*AppComponent.java*) which is your application and if you open the AppComponent via an editor:

```
$ gedit AppComponent.java
```

You will see the ONOS application skeleton with two methods (activate & deactivate) as shown below:



```
package org.hub.app;

import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Deactivate;
import org.apache.felix.scr.annotations.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Skeletal ONOS application component.
 */
@Component(immediate = true)
public class AppComponent {

    private final Logger log = LoggerFactory.getLogger(getClass());

    @Activate
    protected void activate() {
        log.info("Started");
    }

    @Deactivate
    protected void deactivate() {
        log.info("Stopped");
    }

}
```

protected void activate(): Here goes whatever you want to do when the application is activated

protected void deactivate() : Here goes whatever you want to do when the application is deactivated

We want the application to print "*hello world*" once the application is activated, and "*Goodbye*" once the application is deactivated.

1. Simply place the following line into the **`activate()`** method:

    ```
    log.info("hello World");
    ```

2. Place the following line into the **`deactivate()`** method:

   ```
   log.info("Goodbye");
   ```

Note: With ONOS, you cannot use `System.out.print`, instead use the **`log.info()`, `log.warn()`, `log.debug()`, `log.error()`** methods to print.

3. You need to compile the application after any change using the command from **step 4.** In case you receive an error after you modify your application and if the error is related to the test files, you need to delete the test file by simply going back to your application's root folder i.e. the hub folder and then to `/src` folder using:

```
$cd src
```

And you will see a test folder which is recommended to delete. Your application is now ready to be installed on ONOS

Exercise 2 (Running the Hub app in ONOS)

In order to install your application, you have to start the ONOS first. Open a new terminal and run ONOS as we learned earlier, which you have to be in ONOS_ROOT folder (~/onos in SDN-Hub VM) and run the following:

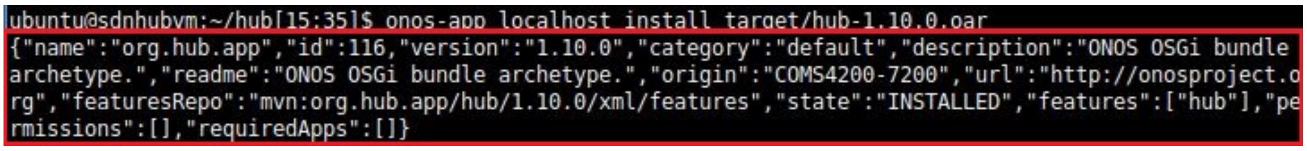
```
$ onos-buck run onos-local -- clean
```

Now, in another new terminal, navigate to the hub folder that we created in the first exercise (e.g.

`$cd  /home/ubuntu/Applications/hub`) and install your app into ONOS via the following command:

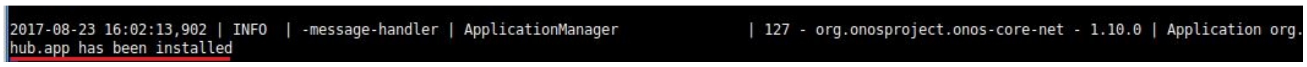
```
 $ onos-app localhost install target/hub-1.10.0.oar
```

As result, you will see a JSON file as shown below



```
ubuntu@sdnhubvm:~/hub[15:35]$ onos-app localhost install target/hub-1.10.0.oar
{"name":"org.hub.app","id":116,"version":"1.10.0","category":"default","description":"ONOS OSGi bundle
archetype.","readme":"ONOS OSGi bundle archetype.","origin":"COMS4200-7200","url":"http://onosproject.o
rg","featuresRepo":"mvn:org.hub.app/hub/1.10.0/xml/features","state":"INSTALLED","features":["hub"],"pe
rmissions":[],"requiredApps":[]}
```

Note: If you check the terminal where you have ONOS running, you should see that the app is being installed as below:



```
2017-08-23 16:02:13,902 | INFO  | -message-handler | ApplicationManager           | 127 - org.onosproject.onos-core-net - 1.10.0 | Application org.
hub.app has been installed
```

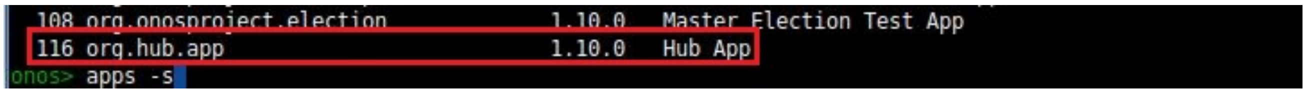
The hub app is now installed on ONOS and you can check this through ONOS CLI. Open a new terminal and access the *ONOS CLI*

```
$ onos localhost
```

And through the CLI, run:

```
onos>apps -s
```

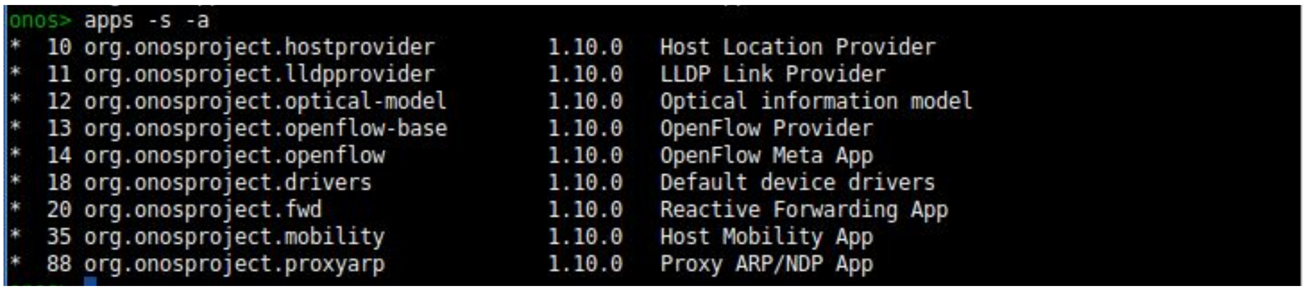
You will see the app installed as shown below:



```
 108 org.onosproject.election            1.10.0   Master Election Test App
 116 org.hub.app                         1.10.0   Hub App
onos> apps -s
```

Note: The hub app is not active yet and you can check active apps via the following commands:

```
onos>apps -s -a
```

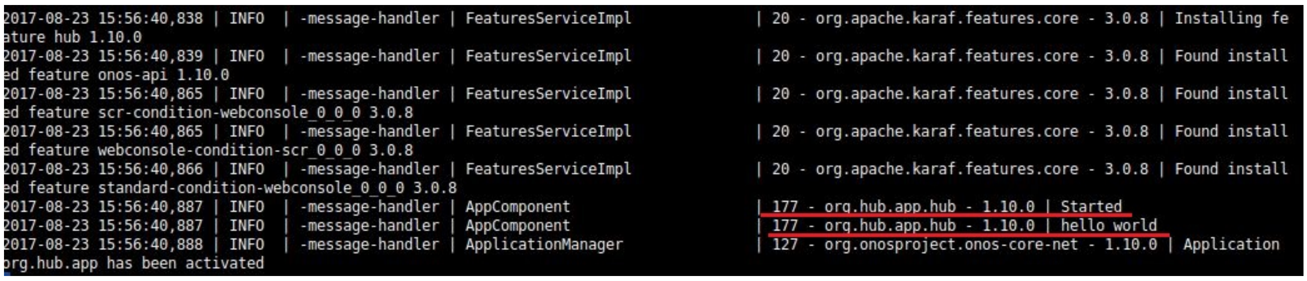


```
onos> apps -s -a
*  10 org.onosproject.hostprovider        1.10.0   Host Location Provider
*  11 org.onosproject.lldpprovider        1.10.0   LLDP Link Provider
*  12 org.onosproject.optical-model       1.10.0   Optical information model
*  13 org.onosproject.openflow-base       1.10.0   OpenFlow Provider
*  14 org.onosproject.openflow            1.10.0   OpenFlow Meta App
*  18 org.onosproject.drivers             1.10.0   Default device drivers
*  20 org.onosproject.fwd                 1.10.0   Reactive Forwarding App
*  35 org.onosproject.mobility            1.10.0   Host Mobility App
*  88 org.onosproject.proxyarp            1.10.0   Proxy ARP/NDP App
```

As we can see our app is not active and to activate the app we need to run the following command from the CLI:

```
onos>app activiate org.hub.app
```

after activating the app, go back to the terminal where ONOS is running and you should see the "Hello World" message that we asked to be printed upon the activation as below:



```
2017-08-23 15:56:40,838 | INFO  | -message-handler | FeaturesServiceImpl              | 20 - org.apache.karaf.features.core - 3.0.8 | Installing fe
ature hub 1.10.0
2017-08-23 15:56:40,839 | INFO  | -message-handler | FeaturesServiceImpl              | 20 - org.apache.karaf.features.core - 3.0.8 | Found install
ed feature onos-api 1.10.0
2017-08-23 15:56:40,865 | INFO  | -message-handler | FeaturesServiceImpl              | 20 - org.apache.karaf.features.core - 3.0.8 | Found install
ed feature scr-condition-webconsole_0_0_0 3.0.8
2017-08-23 15:56:40,865 | INFO  | -message-handler | FeaturesServiceImpl              | 20 - org.apache.karaf.features.core - 3.0.8 | Found install
ed feature webconsole-condition-scr_0_0_0 3.0.8
2017-08-23 15:56:40,866 | INFO  | -message-handler | FeaturesServiceImpl              | 20 - org.apache.karaf.features.core - 3.0.8 | Found install
ed feature standard-condition-webconsole_0_0_0 3.0.8
2017-08-23 15:56:40,887 | INFO  | -message-handler | AppComponent                     | 177 - org.hub.app.hub - 1.10.0 | Started
2017-08-23 15:56:40,887 | INFO  | -message-handler | AppComponent                     | 177 - org.hub.app.hub - 1.10.0 | hello world
2017-08-23 15:56:40,888 | INFO  | -message-handler | ApplicationManager               | 127 - org.onosproject.onos-core-net - 1.10.0 | Application
org.hub.app has been activated
```

Note: Another way to activate the app along with the installation is by placing (!) after "install", which looks like:

```
$ onos-app localhost install! target/hub-1.10.0.oar
```

In this case, the app will be installed and activated simultaneously.

Note (Build and Reinstall): Every time you change/update the code, you need to *build*, *reinstall* (*Not install* because you will get a message saying that the app is already installed) and activate the app. It means you need to repeat **Step 4 from the first exercise and activate the app as we saw in the exercise 2**) or simply run this:

```
$ mvn clean install && onos-app localhost reinstall! target/hub-1.10.0.oar
```

Exercise 3 (Reactive Hub)

The reactive hub basically turns switches into simple hubs, which means each switch simply floods all packets it receives. The hub application we are creating (as the code below) is implemented in a proactive manner i.e. as soon as the app is installed into the ONOS cluster, the controller installs a rule that tells the switch to send all incoming packets out on all ports, except the ingress port.

The code of the reactive hub app is provided below. To run it, copy the code and paste it into the hub app that we created earlier (*AppComponent.java* in the folder *hub/src/main/java/org/hub/app*).

```java
package org.hub.app;

import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Deactivate;
import org.apache.felix.scr.annotations.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


//#################### New Imports  ######################


// In order to use reference and deal with the core service
// For step # 1
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.ReferenceCardinality;
import org.onosproject.core.CoreService;
import org.onosproject.core.ApplicationId;



// In order to register for the pkt-in event
// For step # 2
import org.onosproject.net.packet.PacketProcessor;
import org.onosproject.net.packet.PacketContext;
import org.onosproject.net.packet.PacketService;


// In order you install matching rules (traffic selector)
// For step # 3
import org.onosproject.net.flow.DefaultTrafficSelector;
import org.onosproject.net.flow.TrafficSelector;


// In order to get access to packet header
// For step # 4
import org.onlab.packet.Ethernet;
import org.onosproject.net.packet.PacketPriority;
import java.util.Optional;



//For step #5
import org.onosproject.net.PortNumber;


//#########################################################
```

```java

/**
 * Skeletal ONOS application component.
 */
@Component(immediate = true)
public class AppComponent {

    private final Logger log = LoggerFactory.getLogger(getClass());

//################ Instantiates the relevant services ################
    // You need to refer to the interface class in order to register your
component
    @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
    protected CoreService coreService;
    //  In order to add/delete matching rules of the selector
    @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
    protected PacketService packetService;

    //  To process incoming packets and use PacketProcess methods
   //  such as addProcessor and removeProcessor
    PacketProcessor pktprocess = new Hub();
    private ApplicationId appId;


    @Activate
    protected void activate() {
//   2#-------> You need to register your component at the core
        appId = coreService.registerApplication("org.hub.app"); //equal to
the name shown in pom.xml file

//   3#-------> This is to add a listener for the pkt-in event with
priority
        packetService.addProcessor(pktprocess,
PacketProcessor.director(1));


//   4#-------> This is to add matching rules on incoming packets
        TrafficSelector.Builder selector =
DefaultTrafficSelector.builder();
//    For example, here we just want ARP and IPv4 packets to be forwarded
to this app
        selector.matchEthType(Ethernet.TYPE_ARP);
        packetService.requestPackets(selector.build(),
PacketPriority.REACTIVE, appId, Optional.empty());
        selector.matchEthType(Ethernet.TYPE_IPV4);
        packetService.requestPackets(selector.build(),
PacketPriority.REACTIVE, appId, Optional.empty());




        log.info("Started");


    }



//  5#-------> Override the packetProcessor class in order to change
whatever methods you like
    private class Hub implements PacketProcessor {
        @Override
```

We will provide more details about the code later, but for now we want to run the hub app as we learned before. You need to go back to the main folder of the app e.g. /hub where the pom.xml file is located and recompile the app by running

```
$ mvn clean install
```

Note: again if you receive build failure, in the case you did not remove the test folder as explained above, you can do it now.

Now, to see your app's functionality, you need to deactivate the default ONOS forwarding application (fwd) as well as (proxyarp). From the ONOS CLI, run the following commands:

```
onos> app deactivate org.onosproject.fwd
onos> app deactivate org.onosproject.proxyarp
```

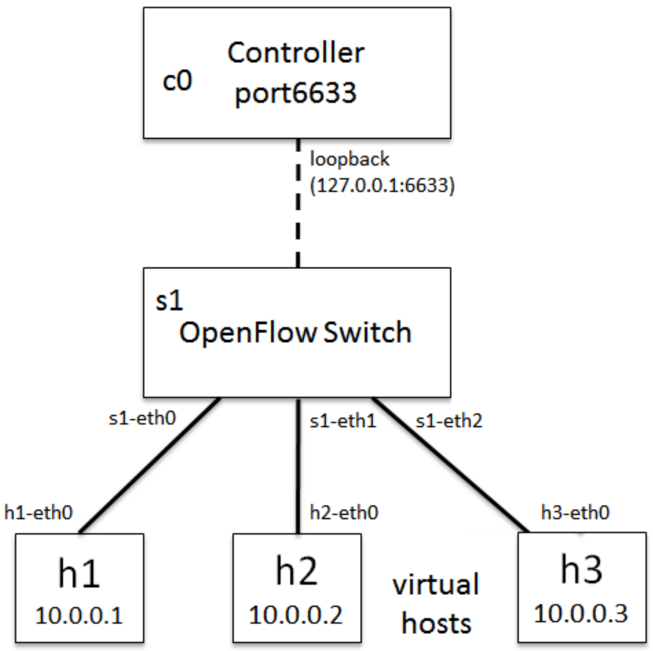
In another terminal, you can now start up Mininet:

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

This tells Mininet to start up a 3-host, single-(openvSwitch-based) switch topology, and set the MAC address of each host equal to its IP address, as shown below.

In order to get sure if there is no other ONOS app which might handle packets, run ping command in Mininet and see that it shouldn't work:

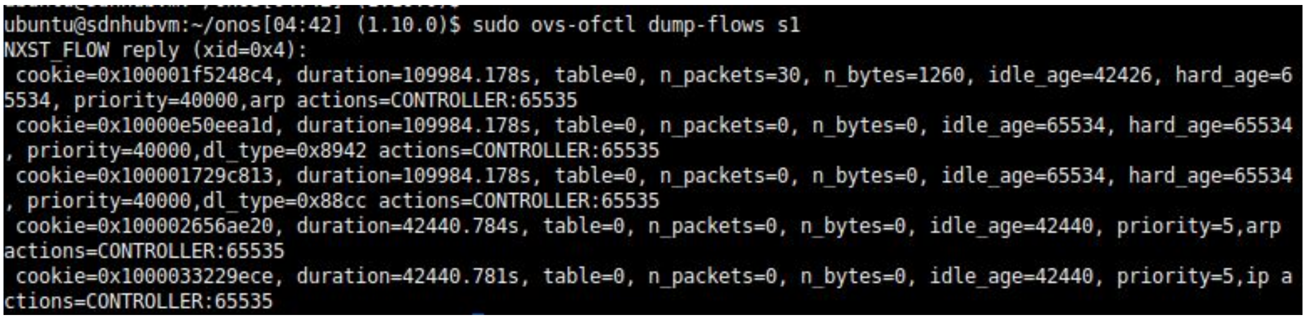
```
mininet> h1 ping -c 1 h2
```



In another terminal, run the following command to check the flow table of the switch:

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

You should see those rules, which are installed by the ONOS core.



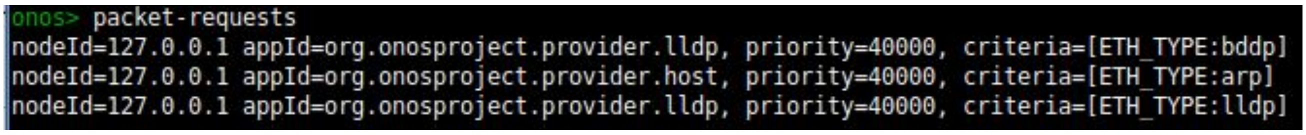
```
ubuntu@sdnhubvm:~/onos[04:42] (1.10.0)$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x100001f5248c4, duration=109984.178s, table=0, n_packets=30, n_bytes=1260, idle_age=42426, hard_age=6
5534, priority=40000,arp actions=CONTROLLER:65535
 cookie=0x10000e50eea1d, duration=109984.178s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534
, priority=40000,dl_type=0x8942 actions=CONTROLLER:65535
 cookie=0x100001729c813, duration=109984.178s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534
, priority=40000,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x100002656ae20, duration=42440.784s, table=0, n_packets=0, n_bytes=0, idle_age=42440, priority=5,arp
actions=CONTROLLER:65535
 cookie=0x1000033229ece, duration=42440.781s, table=0, n_packets=0, n_bytes=0, idle_age=42440, priority=5,ip a
ctions=CONTROLLER:65535
```

The first rule is matching on ***ARP*** with prirotiy set to 40000 for ***the hostprovider app***. The second and the third rules are matching ***bddp*** and ***lldp*** respectively with prirotiy set to 40000 for the **lldpprovider app**.

To check which apps are requesting those packets, i.e. have installed these rules, you can simply go to the ONOS CLI and run:

```
onos>packet-requests
```

And you will see:

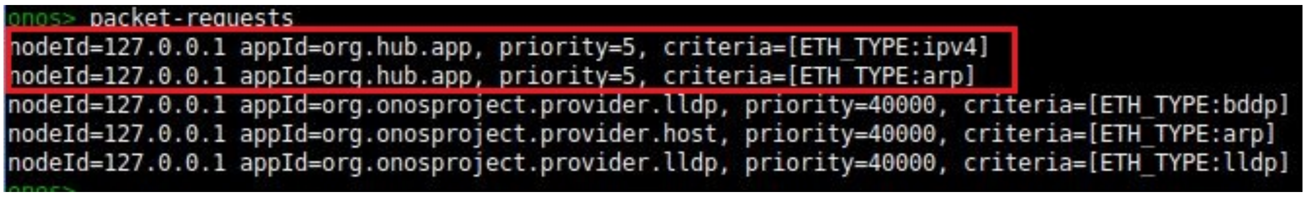


```
onos> packet-requests
nodeId=127.0.0.1 appId=org.onosproject.provider.lldp, priority=40000, criteria=[ETH_TYPE:bddp]
nodeId=127.0.0.1 appId=org.onosproject.provider.host, priority=40000, criteria=[ETH_TYPE:arp]
nodeId=127.0.0.1 appId=org.onosproject.provider.lldp, priority=40000, criteria=[ETH_TYPE:lldp]
```

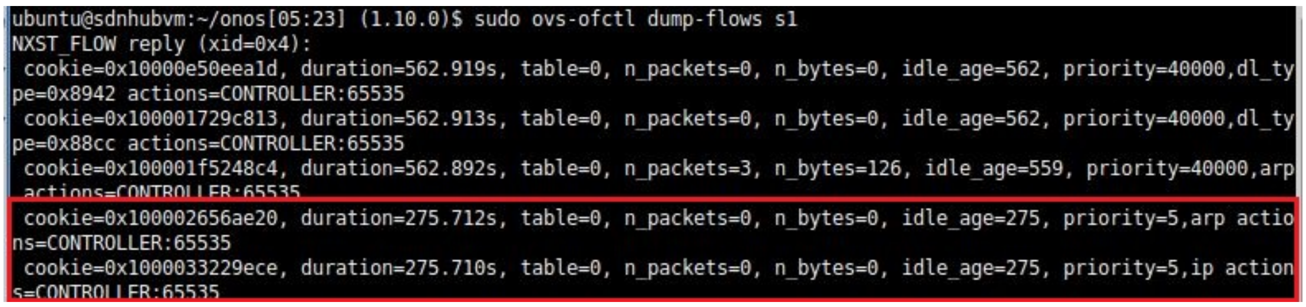
Now you are ready to use your app, so you need re-install and activate it. You need to be inside your app folder (e.g. /hub) and run the following command:

```
$onos-app localhost reinstall! target/hub-1.10.0.oar
```

If you go back and check the rules as well as the packet-requests you will see the difference as shown below:



```
onos> packet-requests
nodeId=127.0.0.1 appId=org.hub.app, priority=5, criteria=[ETH_TYPE:ipv4]
nodeId=127.0.0.1 appId=org.hub.app, priority=5, criteria=[ETH_TYPE:arp]
nodeId=127.0.0.1 appId=org.onosproject.provider.lldp, priority=40000, criteria=[ETH_TYPE:bddp]
nodeId=127.0.0.1 appId=org.onosproject.provider.host, priority=40000, criteria=[ETH_TYPE:arp]
nodeId=127.0.0.1 appId=org.onosproject.provider.lldp, priority=40000, criteria=[ETH_TYPE:lldp]
```



```
ubuntu@sdnhubvm:~/onos[05:23] (1.10.0)$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x10000e50eea1d, duration=562.919s, table=0, n_packets=0, n_bytes=0, idle_age=562, priority=40000,dl_ty
pe=0x8942 actions=CONTROLLER:65535
 cookie=0x100001729c813, duration=562.913s, table=0, n_packets=0, n_bytes=0, idle_age=562, priority=40000,dl_ty
pe=0x88cc actions=CONTROLLER:65535
 cookie=0x100001f5248c4, duration=562.892s, table=0, n_packets=3, n_bytes=126, idle_age=559, priority=40000,arp
 actions=CONTROLLER:65535
 cookie=0x100002656ae20, duration=275.712s, table=0, n_packets=0, n_bytes=0, idle_age=275, priority=5,arp actio
ns=CONTROLLER:65535
 cookie=0x1000033229ece, duration=275.710s, table=0, n_packets=0, n_bytes=0, idle_age=275, priority=5,ip action
s=CONTROLLER:65535
```

Now ping h2 from h1 and this should work:

```
mininet> h1 ping -c 1 h2
```

A. The ONOS Hub structure (Explanation of the code):
For the following discussion, which highlights some of the key parts of the code, it is best if you have the hub source code in front of you. For the full details, please refer to the ONOS API (http://api.onosproject.org/1.10.0/overview-summary.html).

First step, Register ONOS App:
After you create the app skeleton, you need to register your app in the ONOS core, which is responsible for presenting a logically centralized view of network state and logically centralized access to network control functions. This makes it possible to use the services that the core provides. To do this, you should import the following classes:

```
import org.onosproject.core.CoreService;
import org.onosproject.core.ApplicationId;
```

In order to interact with the core system, you need to create a reference of the interface class for using the *constructors*, *methods*, *fields*, *parameters* and *variables* of the class via the following:

```
@Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
protected CoreService coreService;
```

Now, we can use all the coreService methods such as:

1. `getAppId(Short id)`: Returns an existing application id from a given id.
2. `getAppId(String name)`: Returns an existing application id from a given id.
3. `getAppIds()`: Returns the set of currently registered application identifiers.
4. `getIdGenerator(String topic)`: Returns an id generator for a given topic
5. `registerApplication(String name)`: Registers a new application by its name, which is expected to follow the reverse DNS convention
6. `registerApplication(String name, Runnable preDeactivate)`: Registers a new application by its name, which is expected to follow the reverse DNS convention.
7. `version()`: Returns the product version.

For this prac, we only need the *registerApplicaiton* method to register our application as done in line 73.

```
appId = coreService.registerApplication("org.hub.app");
```

Since the *appId* type is an application ID, we need to define it as Application ID before using it as we did in line 67:

```
private ApplicationId appId;
```

However, before we use the *ApplicationId*, we need to import it as we did in line 20.

Building Listenters:
After registering in ONOS core, we can use its services. One of the main services used in any application is subscription to the packet-in events. In order to be notified when the controller receives an Openflow packet-In, we need to create a packet selector that speficies which packet type is needed by the application. You need to import (*PacketService class*) and add a reference to it as in line (27 and 61) following:

```
 import org.onosproject.net.packet.PacketService;


 @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
 protected PacketService packetService;
```

Now, we can use all the *packetService* methods such as:

1. `addProcessor(PacketProcessor processor, int priority):` Adds the specified processor to the list of packet processors.
2. `requestPackets(TrafficSelector selector, PacketPriority priority, ApplicationId appId):` Requests that packets matching the given selector are punted from the dataplane to the controller
3. `requestPackets(TrafficSelector selector, PacketPriority priority, ApplicationId appId, Optional<DeviceId> deviceId):`Requests that packets matching the given selector are punted from the dataplane to the controller.
4. `cancelPackets(TrafficSelector selector, PacketPriority priority, ApplicationId appId):` Cancels previous packet requests for packets matching the given selector to be punted from the dataplane to the controller.
5. `cancelPackets(TrafficSelector selector, PacketPriority priority, ApplicationId appId, Optional<DeviceId> deviceId):` Cancels previous packet requests for packets matching the given selector to be punted from the dataplane to the controller.
6. `emit(OutboundPacket packet):` Emits the specified outbound packet onto the network.
7. `getProcessors():` Returns priority bindings of all registered packet processor entries.
8. `getRequests():` Returns list of all existing requests ordered by priority.

At line # 77 in hub, we used the first method (`addProcessor`) which adds the application to Packet Manager with priority set to (1). The Packet Manager bacisally iterates sequentially through all registered applications and dispatches the packet-In event based on the applications priority. The application with higher priority (larger number) gets the packet earlier.

Building selector:
After adding the listener, we need to tell the core which traffic that our application wants to be notified of when the switch sends it to the controller. It's not ideal to match all traffic coming from the switch, so we need to build the selector. Here, we use the second method (`requestPackets`) with (`TrafficSelector`) interface to specify the matching fields, which simply translated as an OpenFlow rule installed at the switch with action `output:CONTROLLER` sent to the controller.

We need to import the following classes as in line# 31 and 32:

```
import org.onosproject.net.flow.DefaultTrafficSelector;

import org.onosproject.net.flow.TrafficSelector;
```

In line # 81, we create an object for the `DefultTrafficSelector` interface and then we can match on any fields that are supported via the OpenFlow protocol. In our case we match on:

`Ethernet.TYPE_ARP` and `Ethernet.TYPE_IPV4` in line # 83 and 86 respectively.

Refer to this link in order to understand other available fields:
http://api.onosproject.org/1.10.0/org/onosproject/net/flow/TrafficSelector.Builder.html

In the selector we need to specify the packet priority as we used in line # 84 and 87. We imported the packet priority as following:

```
import org.onosproject.net.packet.PacketPriority;
```

Using this class, we can set the packet priority or use existing ones such as:
CONTROL: which is defined as equal to ***40000*** as the *High priority* for control traffic, which results in all traffic matching the selector being sent to the controller.
REACTIVE: which is defined as equal to ***5*** as the *Low priority* for reactive applications

Note: The packet priority is totally different from the listener priority. The packet priority (*set to REACTIVE in line# 84 & 87*) basically tells the switch which traffic selector (forwarding rules) should be executed first and when the traffic is sent from the switch to the controller, Packet Manger determines which application should get the traffic first based on the listener priority (*set to 1 in line #77*).

Packet Processor:
We need this class in order to process the packet after our application receives the traffic that we selected. Therefore, we imported the following classes:

```
import org.onosproject.net.packet.PacketProcessor;

import org.onosproject.net.packet.PacketContext;
```

Then we defined (`pktprocess` in line#66) as a new interface of PacketProcessor

```
PacketProcessor pktprocess = new Hub();
```

in order to override and implement the packet processing as we want to implement it in line #101, which in our case is to set the port to flood and send the packet back to the switch. For this we used PacketContext in order to treat the packet and imported the class:

```
import org.onosproject.net.packet.PacketContext;
```

Refer to the link in order to check other options that you can change in the packet content: http://api.onosproject.org/1.10.0/org/onosproject/net/packet/PacketContext.html

In line #104 we just set the port to flood and send the packet back using `.send()` method

```
pktIn.treatmentBuilder().setOutput(PortNumber.FLOOD);
pktIn.send();
```

Note: In some cases, you might need to create the packet out without receiving packet in, so you can't use the `PacketContext.send()` method (as used in this code in line# 104), which means take the packet and turn it into packet out. In this case you can send a packet out without receiving any packet in via (`PacketService#emit(OutboundPacket)`) which is used to send any packet out any port in the network, including packets that you construct yourself in your application.

From the `PacketContext` class (http://api.onosproject.org/1.10.0/org/onosproject/net/packet /PacketContext.html), you can use the following method to check if the packet is already handled by other application, so you can block the packet from further process by other application.

1. `block()`: Blocks the outbound packet from being sent from this point onward.
2. `isHandled()`: Indicates whether the outbound packet is handled
3. `inPacket()`: Returns the inbound packet being processed

In the `inPacket()` method, there are useful methods for processing the packet and getting information such as where the packet is received from, device ID, source and MAC address ...etc., which we will see in the L2 learning example (Exercise 4) later in this prac.

For example:

`parsed()` Returns the parsed form of the packet, which allows you then to extract information from the packets (http://api.onosproject.org/1.10.0/org/onlab/packet/Ethernet.html).

It's recommended to cancel the packet selector and all listeners upon deactivating the application as we did in the deactivate method (line#112 to line#138) in order to delete the selector and listener from the switch and ONOS core (the method is `packetService.cancelPackets()`).

(**Note**: Before going to the next example, make sure to deactivate your application, i.e. the hub app and kill Mininet)

Exercise 4 (Learning Switch)

Here, we aim to create a Learning Switch and provide a more detailed example of using ONOS. First, we provide a brief summary of the functionality and algorithm of a learning switch:

Initially, the flow table of the learning switch is empty. When a packet arrives, the switch sends the packet via a PacketIn message to ONOS core. The core looks at which application has the highest priority, the packet then is sent to the l2 learning application which extracts information such the source MAC address of the packet, the corresponding switch port it was received on and switch id. It then adds this information to a table. If the MAC address is not known to the controller, it will tell the switch to *FLOOD* it. If the MAC address is already known, the application sets the output port to the corresponding port instead of *FLOOD* and then sends the packet along with installing a rule on the switch, saying that all packets to this address should be forwarded to the corresponding port. (This is how a traditional Ethernet learning switch would work.

The source code of our learning switch ONOS component is provided below. Create a new empty app as explained in exercise 1 and name it l2_learning. Double click on the code box below and then copy and paste the entire code into your java file located at (`l2_learning/src/main/java/org/ l2_learning/app/AppComponent.java`). As a backup, we provided the java file along with this document.

```
package org.l2_learning.app;


import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Deactivate;
import org.apache.felix.scr.annotations.Service;
import org.onlab.packet.IpAddress;
import org.onlab.packet.MacAddress;
import org.onosproject.net.Device;
import org.onosproject.net.flow.DefaultFlowRule;
import org.onosproject.net.flow.DefaultTrafficTreatment;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.packet.OutboundPacket;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;



//#################### Imported those #################################

// In order to use reference and deal with the core service
// For step # 1
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.ReferenceCardinality;
import org.onosproject.core.CoreService;
import org.onosproject.net.packet.PacketService;
import org.onosproject.core.ApplicationId;

// In order to register at for the pkt-in event
// For step # 2
import org.onosproject.net.packet.PacketProcessor;
import org.onosproject.net.packet.PacketContext;

// In order you install matching rules (traffic selector)
// For step # 3
import org.onosproject.net.flow.DefaultTrafficSelector;
import org.onosproject.net.flow.TrafficSelector;
import org.onosproject.net.flow.FlowRuleService;
// In order to get access to packet header
// For step # 4
import org.onlab.packet.Ethernet;
import org.onosproject.net.packet.PacketPriority;
import java.util.Optional;

// In order to access the Pkt-In header
//For step #5
import org.onosproject.net.PortNumber;
import org.onosproject.net.packet.InboundPacket;
import org.onosproject.net.packet.OutboundPacket;
import org.onlab.packet.IPv4;
import org.onlab.packet.Ip4Address;
import org.onlab.packet.IpAddress;
import org.onlab.packet.ARP;
import org.onosproject.net.flowobjective.DefaultForwardingObjective;
import org.onosproject.net.flowobjective.FlowObjectiveService;
import org.onosproject.net.flowobjective.ForwardingObjective;
import org.onosproject.net.DeviceId;
import org.onosproject.net.ConnectPoint;
import java.util.Map;
```

When you are done, you need to build and install your app into ONOS as we learned before. Then, deactivate (`fwd` and `proxyarp` apps) via the ONOS CLI as we learned before.

In another terminal, start up Mininet to connect to the “remote” controller:

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

In yet another terminal, have a look at the flow table:

```
$ sudo dpctl dump-flows s1
```

As before, you should see those rules, which are installed by the ONOS core and there are no other forwarding rules installed.

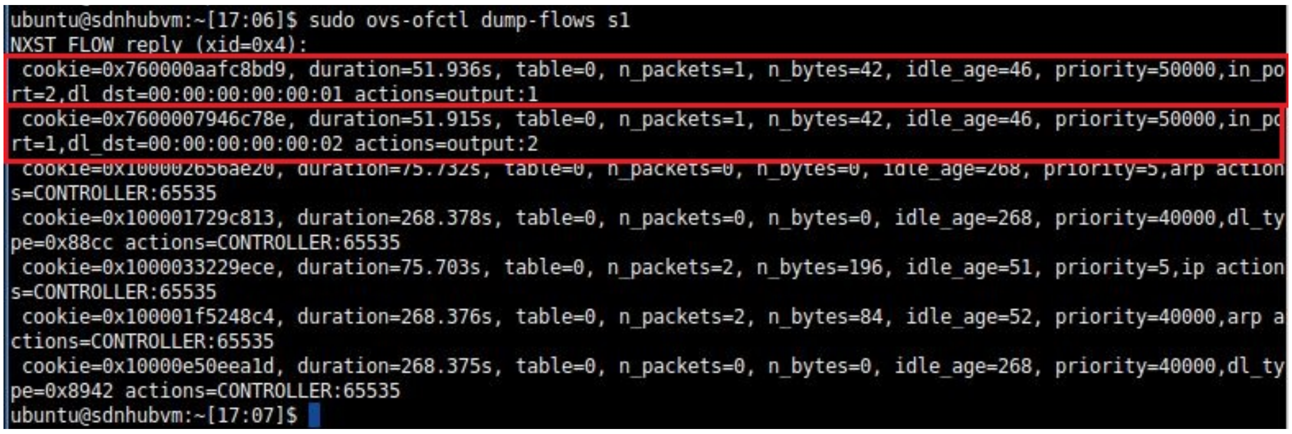
Now run:

```
mininet> h1 ping -c 1 h2
```

If you have deactivated the `fwd` forwarding application in ONOS, ping should not work.

Now, activate your app (l2_learning) and run ping again. Check the forwarding rules that have been installe by the l2_learning app as a result of the ping:

```
$ sudo dpctl dump-flows s1
```



```
ubuntu@sdnhubvm:~[17:06]$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x760000aafc8bd9, duration=51.936s, table=0, n_packets=1, n_bytes=42, idle_age=46, priority=50000,in_po
rt=2,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x7600007946c78e, duration=51.915s, table=0, n_packets=1, n_bytes=42, idle_age=46, priority=50000,in_po
rt=1,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x100002656ae20, duration=75.732s, table=0, n_packets=0, n_bytes=0, idle_age=268, priority=5,arp action
s=CONTROLLER:65535
 cookie=0x100001729c813, duration=268.378s, table=0, n_packets=0, n_bytes=0, idle_age=268, priority=40000,dl_ty
pe=0x88cc actions=CONTROLLER:65535
 cookie=0x1000033229ece, duration=75.703s, table=0, n_packets=2, n_bytes=196, idle_age=51, priority=5,ip action
s=CONTROLLER:65535
 cookie=0x100001f5248c4, duration=268.376s, table=0, n_packets=2, n_bytes=84, idle_age=52, priority=40000,arp a
ctions=CONTROLLER:65535
 cookie=0x10000e50eea1d, duration=268.375s, table=0, n_packets=0, n_bytes=0, idle_age=268, priority=40000,dl_ty
pe=0x8942 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~[17:07]$
```

Try to understand the rules that have been installed, by looking at the output of *dpctl* above, in particular, look at the match fields and the corresponding actions.

You will see that rules are installed not just based on the destination MAC address (as a traditional Ethernet switch would do), but that we install rules matching both source AND destination MAC addresses. Can you explain why this is necessary?

That’s it. The aim of this prac was to give you a basic introduction on how to write and build simple ONOS apps. Obviously, we’ve only scratched the surface and there is much more to learn. A good starting point for exploring this further are the ONOS API Docs:

http://api.onosproject.org/1.10.0/

If are stuck and need help for your project, please don’t hesitate to ask for help.