Intent Based Application

Group Project Report for Advanced Network Architectures And Wireless Systems

Francesco Iemma
Yuri Mazzuoli
Giovanni Menghini

M.Sc. in Computer Engineering

Academic Year 2021/22

Contents

Chapter 1

Implementation

1.1 Floodlight Functionalities

1.1.1 How To Compute A Path: Routing Service

1.1.2 How To Install A Path: Forwarding Base

1.2 Floodlight Extended Forwarding Module

The main module of our package is IntentForwarding. This module is an extension of the floodlight forwarding module[1], in general the added functionalities are:

- An array list that remembers the list of active intents
- The extension of the method *ProcessPacketInMessage* in order to check if the packet that triggered the packetIn is from two host that are allowed to communicate (an intent between the two exists) or not.

1.2.1 Intent Life Cycle

An intent is created, is active and then, when the time indicated in the intent establishment request finishes, expires.
When an intent is created the following operations are done:

- It is checked if an intent with the same IPs already exists. If yes no other operations are performed.
- A new timeout task is created and scheduled in order to be executed in the amount of time indicated by the intent timeout
- The intent is added to the array list

Notice that the creation of the timer is needed in order to implement the intent expiration and to perform some operations when this happens.
The operations that must be done when the intent expires are:

- Install rules in certain switches in order to deny the communications between the host of the expired intent
- Delete the intent from the intent database

Thus the main problem is to decide which are the switches where we have to install the rules. Let assume that we have two host: *host1* and *host2* and an intent between them exists and it is going to expire. In order to minimize the packets in the network from hosts of an expired intent it was decided to install the rules in the first switch encountered by the *host1* when sends the message to *host2* (we will call this switch *switch1*) and in the first switch encountered when *host2* sends a message to *host1* (we will call this switch *switch2*).
The rules installed on each of the two switches are:

[1] net.floodlightcontroller.forwarding.Forwarding

- Deny IPv4 from *host1* to *host2*
- Deny IPv4 from *host2* to *host1*

Both rules are valid for 5 seconds by default. See figure 1.1 for a graphical representation.

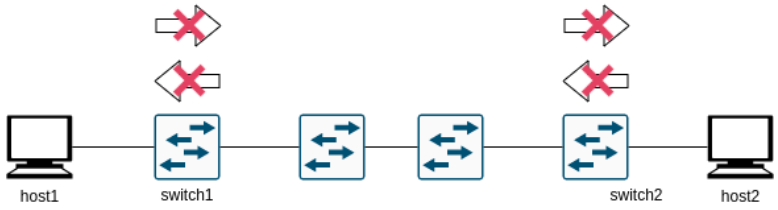

Figure 1.1: Graphical representation of where the rules for denying the communications between two hosts are installed

Thanks to this rules no other packets are allowed to pass from *switch1* and *switch2* after the intent expiry. When the rules are no longer valid for each packet arriving to *switch1* or *switch2* a packetIn is generated and thus the *ProcessPacketInMessage* method will check if that packet is allowed (an intent has been re-established) or not (the old intent has expired and no other intent have been established).
Notice that due to the fact that the rules installed in the switches are valid for a certain time, no intent can be established until that rules are no longer valid. In the default case the rules are valid for 5 seconds and so a new intent regarding one of the two hosts can be established 5 seconds after the expiry of the old intent. That time can be changed by the network manager through REST.
The last problems that need a solution is how we can know which is the identifier of *switch1* and *switch2*. When the timeout task (the task that will be performed when the intent expires) is created we pass to the constructor:

- A reference to the object IntentForwarding, this is necessary in order to call the function who is in charge to install rules in the switches
- A reference to the object representing the intent associated to that timeout task. That object is **HostPair** and it is composed by:
  - IPv4Address host1IP
  - IOFSwitch sw1
  - IPv4Address host2IP
  - IOFSwitch sw2
  - long timeout

  That reference is necessary in order to retrieve the identifier of *switch1* and *switch2*. That identifiers are set when the first packetIn coming from the first switch encountered by the packet sent by an host is handled by the extended forwarding module i.e. IntentForwarding (for more on this see 1.2.2).

Here we can see the code executed when we add an intent.

```
1   public boolean addNewIntent(HostPair newPair) {
2     System.out.print("AddNewIntent Called\n");
3     if(intentsDB.contains(newPair)) {
4       log.info(" Intent already present in Intents List");
5       return false;
6     }
7
8     long timeout = newPair.getTimeout();
9     Timer timer = new Timer();
10    TimerTask task = new TimeoutTask(newPair, this);
11    timer.schedule(task, timeout);
12
13    intentsDB.add(newPair);
14    return true;
15  }
```

1.2.2 IPv4 Management

When a packetIn regarding an IPv4 packet arrives the operations donw are:

- The intents database is searched:
  - If exists an intent between the sender and the receiver (or viceversa) then
    - If the sender is *host1* and *switch1* in the object HostPair representing the intent, is null then *switch1* is set equal to the identifier of the switch that has sent the packetIn that has triggered this execution of ProcessPacketInMessage. Same thing is done for *switch2*.
    - Then the method of the super class (Forwarding.java) is invoked
  - Otherwise two rules are installed on the switch that sent the packetIn in order to deny the communication between the two hosts for a certain amount of time (the default timeout is 5 seconds but that value can be customized through REST API).

  Here a section of the method *processPacketInMessage*. Notice that when setting the switch id is the method *setSw(IOFSwitch switch)* that checks if the switch is already set or not and set the switch only if it is not already set.

```
1   HostPair hp = null;
2   int hpIndex = intentsDB.indexOf(new HostPair(sourceIP, destinIP));
3   if (hpIndex != -1)
4     hp = intentsDB.get(hpIndex);
5
6   if(hp != null && intentsDB.contains(hp)) {
7     System.out.printf("allowing: %s - %s on switch %s \n",
8       sourceIP.toString(), destinIP.toString(), sw.getId());
9     if(hp.getHost1IP().equals(sourceIP))
10      hp.setSw1(sw);
11    if(hp.getHost2IP().equals(sourceIP))
12      hp.setSw2(sw);
13    return super.processPacketInMessage(sw, pi, decision, cntx);
14  }
15  denyRoute(sw, sourceIP, destinIP);
16  denyRoute(sw, destinIP,sourceIP);
17  return Command.CONTINUE;
```

1.2.3 ARP Management

The ARP protocol is used to retrieve the layer 2 address of an host given its IP address. When an ARP packet arrives to a switch a packetIn is sent to the controller and so the processPacketInMessagge is invoked, the method understands that the packet that has triggered the packetIn is an ARP packet and so it must be handled differently. The first operation done is checking if an intent between the sender and the receiver exists using their IPs. If an intent exists then the method of the super class is invoked otherwise the ARP packet is blocked. The rules to avoid communication between the MACs address carried inside the ARP packet are installed only if that packet is an ARP response, in that case the target MAC address is not 00:00:00:00:00:00. In fact installing a rule that deny to send an ARP messagge from a certain mac X to 00:00:00:00:00:00 means deny X to runs the ARP protocol because for each ARP request the target mac is 00:00:00:00:00:00 i.e. unspecified mac.

1.3 REST API For Establishing An Intent

1.4 Responsiveness To Link Failures & Topology Changes

Chapter 2

Testing