
# 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

# Introduction

This project is aimed to design and develop an intent based application. The scenario is the following: *"Consider a set of clients that can communicate through a redundant network (e.g., based on a spine-leaf topology.) An external application can request to install paths between host pairs, by just specifying the endpoint identifiers, we refer to this as an host-2-host intent. The network has to allow communications only among the specified host pairs. Moreover, the network has to automatically reconfigure in case of link failures. Design and realize a system that allows users to request and withdraw host-to-host intents, and configures the underlying network accordingly."*
The objectives of the projects are:

1. *"Implement a Floodlight module that exposes a RESTful interface allowing clients to create/delete host-to-host instents. The module will then dynamically install and update flow rules in the network to allow the communication among specified pairs. Possible path switches must occur transparently to clients."*

2. *"Test the overall system using Mininet and Floodlight, devising proper scenarios to demonstrate the above functionalities."*

# Chapter 1

# Implementation

The objective of the intent based application is mainly to expose a REST interface to allow the request of an **intent**. An **intent** is a request, done by an host, to have a link with another host, this link must tolerate link failures. Hence the controller must implement a module that is in charge of:

- Computing the best path between hostA and hostB
- Installing in the switches of the network the proper rules to establish this path
- Being responsive in case of a link failures and establishing a new path to maintain the link alive

In the implementation of the application we have re-used some Floodlight modules extending some classes where needed. In the following sections we will analyze the choices done and the classes extended in order to implement the features shown previously.

## 1.1 Floodlight Functionalities

### 1.1.1 How To Compute A Path: Routing Service

Floodlight by default exposes a REST inteface to retrieve a path between a source (switch DPID and port ID) and a destination (switch DPID and port ID).
The computation of the path is given from the method *getPath(srcDpid, srcPort, dstDpid, dstPort)* of the class implementing the interface *IRoutingService*[1].
Thus the idea is to reuse this function in order to retrieve the path between hostA and hostB.

### 1.1.2 How To Install A Path: Forwarding Base

Once we have the path we need to install it inside the switches of the network in order to have the path established and working.
To do so Floodlight provides the abstract class ForwardingBase[2] which is in charge of implementing a forwarding module. The forwarding module "is responsible for programming flows to a switch in response to a policy decision". The implementation of the abstract class must implement the following abstract method:

```
1    public abstract Command processPacketInMessage(IOFSwitch sw, OFPacketIn pi,
2    IRoutingDecision decision, FloodlightContext cntx);
```

The Floodlight standard implementation of this abstract class is the class Forwarding[3].

[1] net.floodlightcontroller.routing.IRoutingService
[2] net.floodlightcontroller.routing.ForwardingBase
[3] net.floodlightcontroller.forwarding.Forwarding

## 1.2 Floodlight Extended Forwarding Module

Thus the idea is to extend Forwarding and redefine the method processPacketInMessage which is in charge of defining what the module must do when a packetIn arrives.
In our scenario the operations that must be performed when a packetIn arrives are:

- Retrieving the path using the method *getPath(..)* provided by IRoutingService
- Installing the path using the method *PushRoute(..)* inherited by this module from ForwardingBase

It is important to underline that before doing so, the logic of the method must check if the packetIn arriving is due to the packet coming from the hostA that has asked the intent and that has as destination the hostB indicated in the intent.
To do this check it is needed a data structure that is in charge of maintaining the list of intent asked and established.

## 1.3 REST API For Establishing An Intent

When the hostA wants to send an intent to make a connection with hostB it has to use the REST API. This API is provided by the classes: IntentWebRoutable, AddNewIntent, DelIntent and GetIntent; these classes use methods from IntentForwarding.
When the REST API receives the intent, it establishes it inserting an entry in a dedicated data structure where it is indicated:

- The Source
- The Destination
- Intent Timeout

The possible API's that we can use are:

- "/addNewIntent/json", that is used to insert new intent. This API gets from a JSON file two IPs (representing source and destination) and a timeout. This use the addNewIntent function of the IntentForwarding class that, first check if new intent is already presents, then in case it isn't put it in intentDb that is an ArrayList of HostPair. Moreover, this function also set the timeout time of the pair.
- "/delIntent/json", that takes from a JSON file two IPs and a timeout and removes the intent corresponding to them. To do this, it use the delIntent function of IntentForwarding, that iterates the intentDB list and removes the intent with IPs passed in through JSON.
- "/getIntents/json", that is used to get all intents present in specific moment. It use the getIntents function of IntentForwarding that simply return intentDb.

Then the intent will be pushed in the network when the source host tries to send a packet to the destination host. When this happens a packetIn sending from the first switch that receives the packet will be sent to the controller and will be handled by the forwarding module seen in 1.2

## 1.4 Responsiveness To Link Failures & Topology Changes

To be responsiveness to topology changes we can exploit the ITopologyListener inteface. In fact in the Floodligh documentation for the TopologyService it is written: *"All the information about the current topology is stored in an immutable data structure called the topology instance. If there is any change in the topology, a new instance is created and the topology changed notification message is called. If other modules want to listen for changes in topology they can implement the ITopologyListener interface."*[4].

[4]https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343623/TopologyService+Dev

Thus we can implement a class that implements ITopologyListener and that each time a topology change arrives it recompute the paths of intent involved in the topology change (i.e. the ones that has a link which doesn't work anymore). To do so we can use the TopologyInstance (updated before sending the notification message that the topology has changed) to understand which intent are involved in the topology change.

# Chapter 2

# Testing

## Languages and Frameworks

In order to test our system, we need to use a tool to create a *virtual network* inside our machine; we used `mininet`, exploiting the `python2` APIs. To simulate input from an external application we also used a python2 library, called `requests`, which is able to act like an HTTP client.

## 2.1 Scenario

We want to simumlate a tipical data center scenario with a single LAN, implemented in a *Spine-Leaf* topology. This configuration is widely used thanks to it's easy scalability and sufficent redundancy.

Figure 2.1: an example of *spine-leaf* topology