UNIVERSITY OF PISA

Master in Computer Engineering
Project for Advanced network

Dynamic Service Reallocation using floodlight

Author:
Tommaso Burlon

Academic year
2021/2022

Summary

1 Introduction

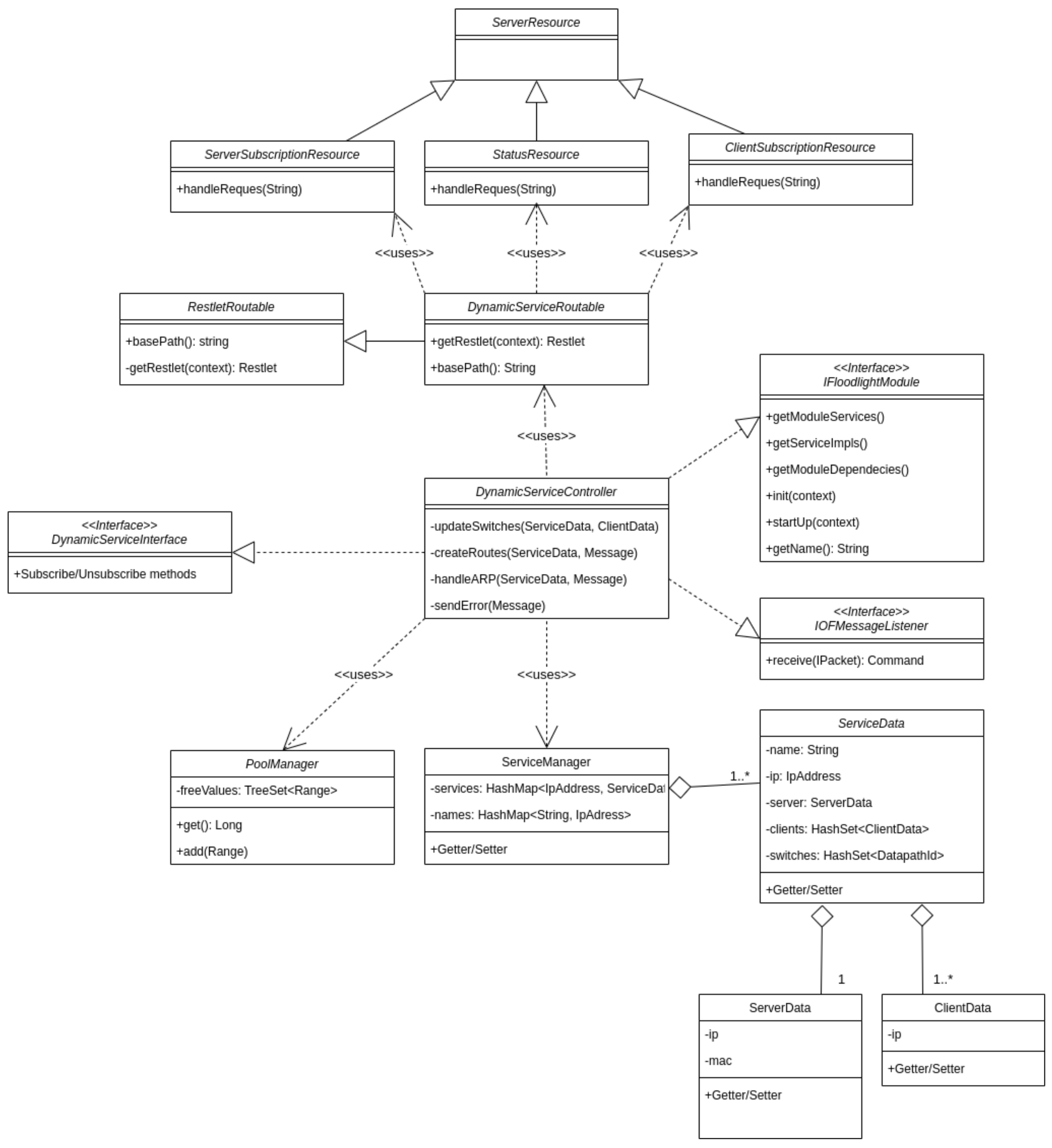
In cloud application it is often useful to implement a dynamic service reallocator. Indeed application in cloud computing should not known the exact location (in terms of IP address for example) of a specific service, in this way different machines could host the same service. This approach is necessary because it offer high scalability and reliability in a cloud system by decoupling a client with a specific service. A lot of cloud orchestrators (e.g. Kubernetes) offer this kind of service. In this project I've implemented a Dynamic service reallocator using floodlight and I tested this module using mininet.

2 Specification

The project requires to implement a floodlight module that programs Open Flow switches to link clients subscribed to a specific service with a server that host that service. To subscribe/unsubcribe a client to a service and to reloacate a specific service to a server a REST interface hosted by the floodlight controller should be implemented. To access a specific service a client should communicate with a particular VIRTUAL_IP.

3 Architecture

The architecture of the floodlight package is shown in the figure with an UML scheme.


ServerResource
ServerSubscriptionResource
+handleReques(String)
StatusResource
+handleReques(String)
ClientSubscriptionResource
+handleReques(String)
<<uses>>
<<uses>>
<<uses>>
RestletRoutable
+basePath(): string
-getRestlet(context): Restlet
DynamicServiceRoutable
+getRestlet(context): Restlet
+basePath(): String
<<Interface>>
IFloodlightModule
+getModuleServices()
+getServiceImpls()
+getModuleDependecies()
+init(context)
+startUp(context)
+getName(): String
<<uses>>
DynamicServiceController
-updateSwitches(ServiceData, ClientData)
-createRoutes(ServiceData, Message)
-handleARP(ServiceData, Message)
-sendError(Message)
<<Interface>>
DynamicServiceInterface
+Subscribe/Unsubscribe methods
<<Interface>>
IOFMessageListener
+receive(IPacket): Command
<<uses>>
<<uses>>
PoolManager
-freeValues: TreeSet<Range>
+get(): Long
+add(Range)
ServiceManager
-services: HashMap<IpAddress, ServiceDat
-names: HashMap<String, IpAdress>
+Getter/Setter
1..*
ServiceData
-name: String
-ip: IpAddress
-server: ServerData
-clients: HashSet<ClientData>
-switches: HashSet<DatapathId>
+Getter/Setter
1
1..*
ServerData
-ip
-mac
+Getter/Setter
ClientData
-ip
+Getter/Setter

3.1 Java Classes

In the package 11 classes have been implemented. 3 classes (ServiceData, Server-Data, ClientData) simply contains the data for a specific object (a service, a server and a client). 4 classes (StatusResource, ClientSubscriptionResource, ServerSubscriptionResource, DynamicServiceRoutable) handle the REST interface for client/server subscription The PoolManager class has been implemented to store all the IP address that the module can use as VIRTUAL_IP for a specific service. The PoolManager offers a O(log(n)) insertion and O(log(n)) extraction with a worst case scenario O(n) memory use. The PoolManager is optimized for memory usage in the average case. The serviceManager is a utility class to handle every service that the module is running. The last concrete class implemented is the DynamicServiceController, this class implement all the logic of the module. This controller handle the messages that arrives from the OVS switches.

4 Example Application

An example application has been implemented to test the floodlight module and the service implementation. For this purpose a client application, a simple REST server and a server application have been implemented.

4.1 Client

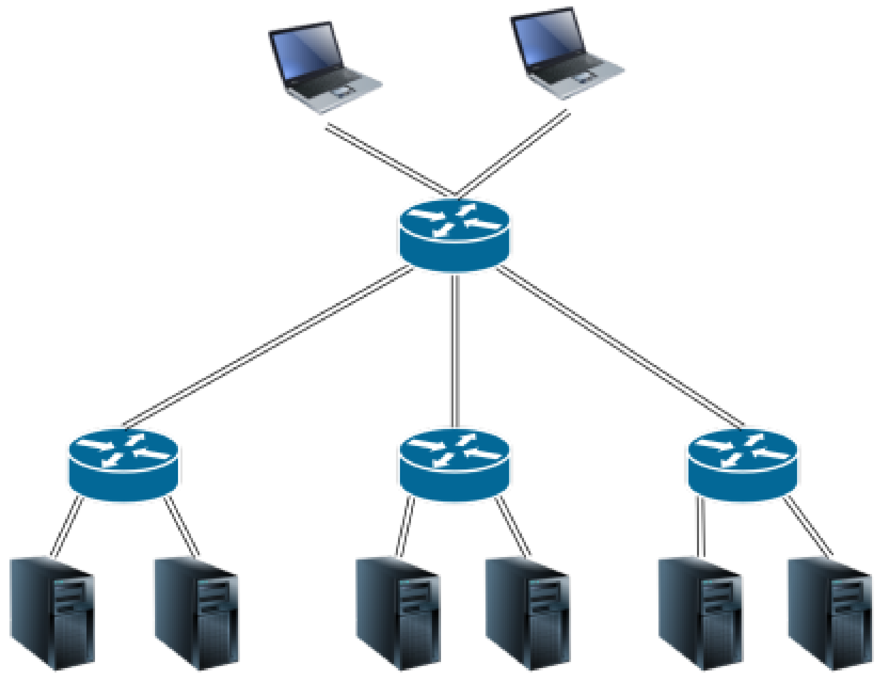
The application developed to test the client can use 4 different command: sub, unsub, get, help. the sub command subscribe a client to a service if this service already exist, if the subscription happens the REST interface should answer with the VIRTUAL_IP where the service is hosted. the unsub command unsubscribe the client to a service. the get client request the service, in these example the service is simply a string containing the IP address of the server that hosts the service. the help command print the instruction of the application.

4.2 Server

The application developed to test the service can use 3 different command: sub, unsub, help. The sub command links the current server to a specific service, if the service does not exists, the floodlight module will create a new service and will associate to that service a particular VIRTUAL_IP. The unsub command will delete the service that the server host. A specific server cannot host two different services (the floodlight module can handle this case if two different IP addresses are used).

5 Testing

To test the floodlight module the mininet program has been used. The testing programs will create the virtual network represented in the following figure.



Then it performs 5 different tests for the main functionality of the systems.

6 User Manual

Insert all the classes inside the floodlight directory into the src folder of floodlight, update the META_INF directory of floodlight to load also the DynamicServiceController. To run floodlight execute into the floodlight directory:

```
$ ant run
```

To test the module in a custom way with the example programs go to the example folder and execute:

```
$ python3 network.py <IP-ADDRESS>
```

where <IP-ADDRESS> is the ip address of the current machine, now the mininet interface should appear. with the virtual network described in the Testing section. The network contains 2 client (called client0, client1) and 6 servers (called server0..5). To execute a specific command run in the mininet CLI:

```
> server0 sub service
> client0 sub service
> client0 get 10.0.0.5
```

To execute the automate test programs run:

```
$ python3 network.py <IP-ADDRESS> test
```