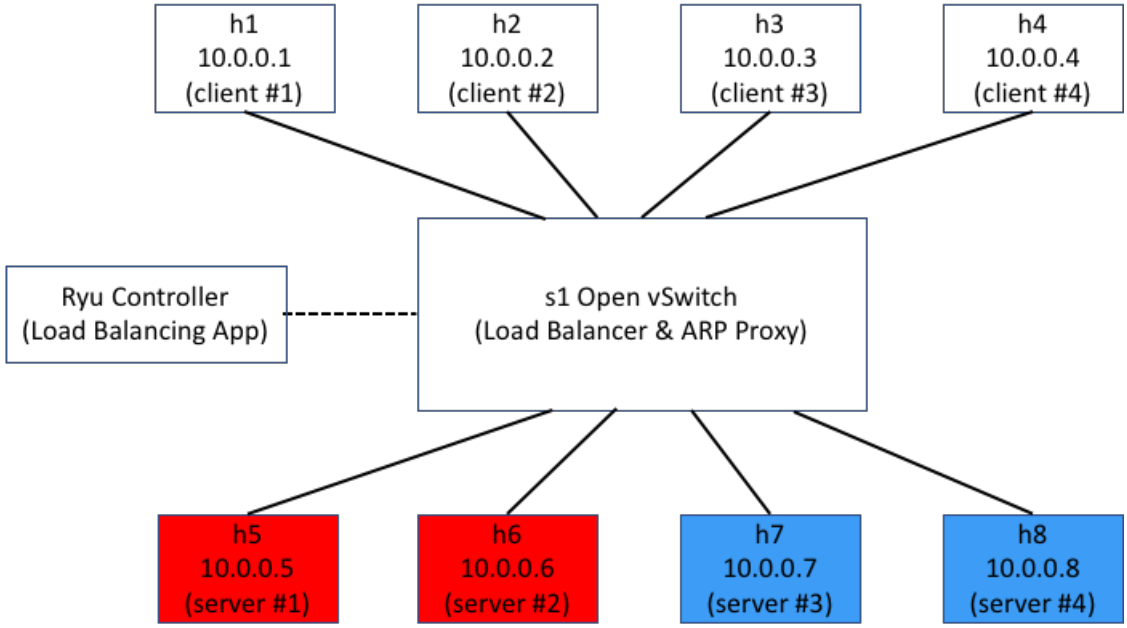
Assignment 1

Load balancing is a classic network-side application that is useful for data centers, ISPs and other enterprise networks. In this assignment, we will focus on a simple load-balancer implementation that balances the load stemming from different end-users, to a server farm, while considering features such as transparency and traffic isolation.

Overview

The network topology you'll use in this assignment includes 8 hosts and a switch with an OpenFlow controller (Ryu)



For more details on Ryu tutorial, see Writing Your Ryu Application. As shown in the figure, the first 4 hosts (h1 to h4) are clients for services offered by the server hosts (h5 to h8). There are two types of services, 'red' and 'blue' and each service is supported by a server farm of 2 nodes. Only the servers from the specified server group could serve the corresponding service requests. For example, if h1 request 'blue' service, it could only be served by h7 or h8. And if all hosts (h1 – h4) request 'blue' service, the flows should be balanced among h7 and h8.

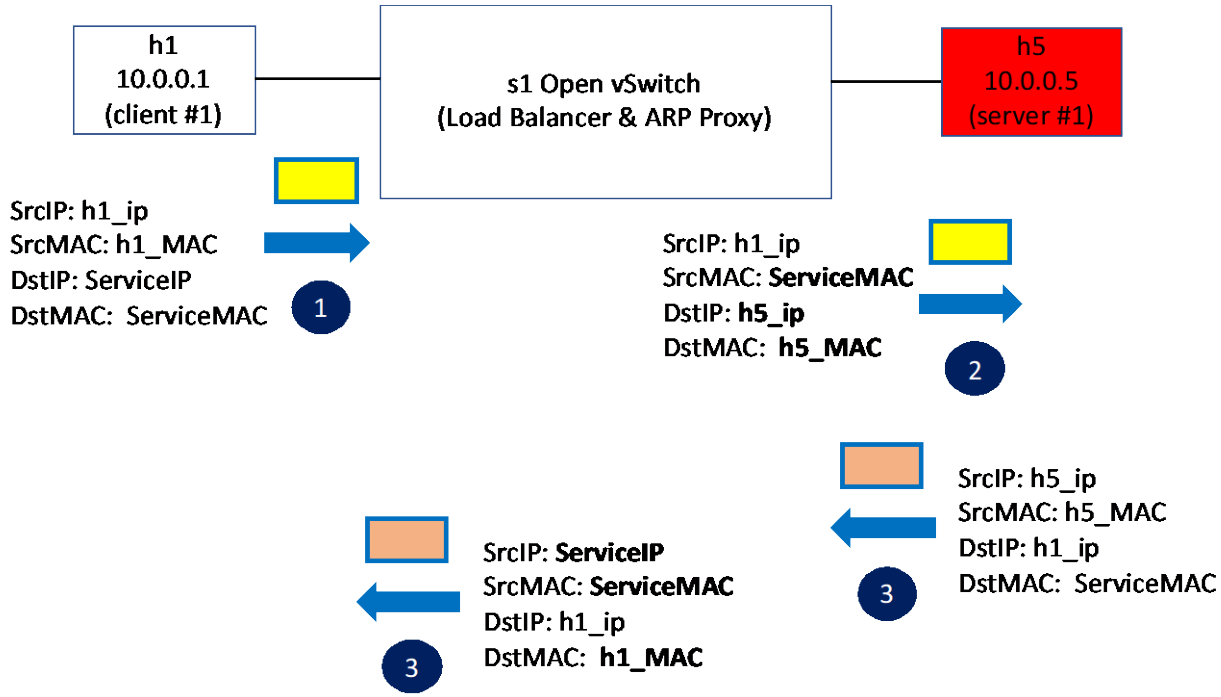
The switch acts as a load balancer to balance the flows from the clients to the servers. It also functions as an ARP proxy for two purposes: (1) allowing clients to learn the mac address (which is the mac address of the load balancer) for its service request traffic and (2) allowing the load balancing application to direct the service response traffic to the appropriate output port.

The load balancer is transparent to all hosts. Therefore, the clients should never learn the real IP and MAC address of the server that it talks to and the server should not be able to learn the real MAC address of the clients either. (But the server should see the real IP address of the client. Otherwise, the load balancer would not know which client the response packets should go to) For

this purpose, there are two public ip addresses that are exposed to the clients, corresponding to two different services, respectively.

Client-Switch Communication for Service Packets

Following is a simplified example to demonstrate how load balancer works. Basically, the control app needs to insert the correct flow entries to make sure the load balancer works properly.



Notice that the ARP handling is omitted from the figure. To handle the ARP requests and responses properly, the switch needs to do the following steps:

- The switch should preemptively ask for the MAC addresses of all the servers with crafted ARP requests, in order to associate these MAC addresses and the corresponding switch ports with the real IP addresses of the servers. This query should be performed upon the connection establishment of the controller to the switch (CONFIG_DISPATCHER state of the switch in Ryu controller).

- To answer the ARP requests from the clients searching the MAC of the service IP address, the switch should relay ARP responses that are sent by the servers with a fake MAC address (which is the ServiceMAC in the figure). It is similar to answer the ARP requests from the servers.

To achieve load balancing, for each new flow from a client, select a server at random from the server farm to. (Or you could implement a round-robin fashion load balancing scheme. Random selection is an approximation that is easier to implement). In the provided code skeleton, the

flow entries for service traffic have a 10-second-idle timeout. Requests of the same service after time out should be considered as new flows.

Understanding the Code

To begin with, download the files for this assignment. Three files are contained under folder `assign1`:

- `lb.py`: a skeleton class which you will update with the logic for load balancing. You can add any functions and include any python packages as you need.
- `lb_config.json`: the configuration file for the load balancing application that needs to be loaded upon the application start
- `setup.sh`: a bash script to install the latest version of openvswitch in your VM. Put this file under the ovs directory after you clone a copy of the ovs repository on your VM. For installation, simply type `` `sudo ./setup.sh` ``.

OVS Installation

Clone OVS repository with command:
`` `git clone https://github.com/openvswitch/ovs.git` ``

Then put `` `setup.sh` `` under folder `ovs`. Running `` `sudo ./setup.sh` `` will automatically start `ovsdb-server` and `ovs-vswitchd`.

Since the default openvswitch and ovsdb-server might already be running, you could stop them using command `` `sudo service openvswitch-switch stop` ``

The `setup.sh` file contains all the commands including boot, configure and compile, which could be commented out for just running the switch without re-compiling. Or you could comment out the last three line to just compile the switch without running it.

Before ovsdb-server itself can be started, configure a database that it can use:

```
$ mkdir -p /usr/local/etc/openvswitch
  $ ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
          vswitchd/vswitch.ovsschema
```

(More details could be found from file `ovs/Document/intro/install/general.rst`)

Ryu OpenFlow v1.3.0

Follow the instructions from https://github.com/osrg/ryu to install Ryu on your VM.

OpenFlow v1.3.0 implemented by Ryu is quite different from OpenFlow v1.0. You could learn the code by reading `ryu/ofproto/ofproto_v1_3.py` and `ryu/ofproto/ofproto_v1_3_parser.py`.

Configuration file `` `lb_config.json` `` should be passed to the load balancing application as an argument. To understand how configuration is managed and used by Ryu, read file `ryu/flags.py` carefully.

Group table feature is supported by OpenFlow v1.3.0 and it potentially makes it much easier to implement load balancing. But it is more like a proactive rather than reactive solution. In this assignment, the controller needs to perform the load balancing operations. You are encouraged to use the group table feature as an exercise and to compare its performance with the reactive solution you implemented.

Most of the functions or mechanisms needed for this app are already implemented in the example applications, tests or utilities of Ryu. The source code of Ryu is the one of best references you could find to learn how to customize applications with Ryu.

Try NOT to modify any other files in Ryu when implementing this application

Mininet Setup

use command `` `sudo mn --topo single,8 --mac --switch ovsk --controller remote` `` to setup your network

you can use `iperf` or `ping` to test your code.

Submission

Submit your code, `lb.py` and any other files if you made modifications to Ryu to Canvas.

DUE: Oct 2nd 11:59 pm ET