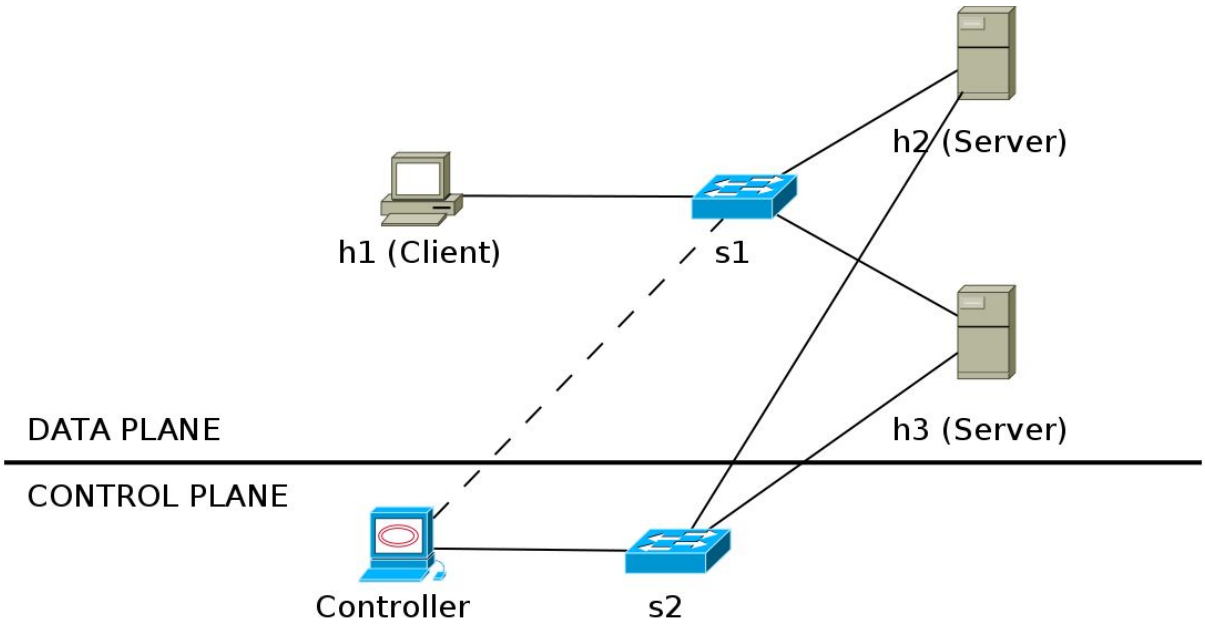
Challenge 2: HTTP load-balancer

Your task is to implement an HTTP load-balancer using OpenFlow.

Overview of the experimental setup



We will be using Mininet to simulate a network consisting of three hosts: a client (h1) and two HTTP servers (h2 and h3). All three hosts are linked to an OpenFlow switch (s1). The OpenFlow switch, working in conjunction with the controller, assigns connections originating from the client to either of the two servers.

In order to query the servers about their load, the controller can reach them directly via a second switch (s2). For all intents and purposes, s2 is a simple Ethernet switch, and it is not under the influence of the controller.

The role of the controller

For each new connection coming from the client, the controller must pick which of the two servers will handle it. After seeing the first packet of the connection (the SYN), the controller must install a flow entry that directs all subsequent packets to the chosen server.

The controller must also *periodically* query the servers on their load and use that information when deciding how to distribute incoming traffic in the immediate future. The server which is under less load should receive more traffic, while the one bearing a heavier load should receive less. (Making a round of queries for each packet that the controller receives is not acceptable.)

The experimental setup in detail

With the exception of the controller, everything is running inside a Mininet VM.

The VM has two network interfaces.

The Mininet topology consists of three hosts and two switches:

- h2 and h3 each house an Apache web server and an ssh server. Both of them have the same IP and MAC address.
-
- s1 links all three hosts and is configured to connect to the controller.
- s2 bridges h2 and h3 and one of the VM's interfaces. It is through that interface that the controller will be able to reach h2 and h3.
-

The Apache servers will be using mod_status.

Setting it all up

- Setup Floodlight on your physical machine.
- Get the Mininet VM.
  - https://github.com/mininet/mininet/wiki/Mininet-VM-Images
  - Add an extra interface to the VM. If you are using VirtualBox, use a Host-only adapter for the second interface (the bridged adapter has some issues).
  - If you are using VirtualBox, enable promiscuous mode on the host-only interface (Settings -> Network -> Adapter 2 -> Anvanced; select "Allow all")
  -
- Setup Apache
  - apt-get install apache2
  - Edit /etc/apache2/mods-available/status.conf. Comment out the line that says "Require local"
  - Set up two extra instances of the daemon:

  sudo bash /usr/share/doc/apache2/examples/setup-instance h2
  sudo bash /usr/share/doc/apache2/examples/setup-instance h3
- Get the mininet script and run it: challenge2.py
  - You'll need to edit the controller's IP and the name of the second network interface (the one which will be bridged with h2 and h3).

Hints

- You only need to concern yourself with TCP traffic.
- **EDIT**(22/11): ARP can be easily dealt with: you can add static entries on each of the hosts (use "arp -s") or insert a flow mod that floods all packets with ethertype 0x0806. (The latter is preferable.)
-
- Try to minimize the number of flow mods you send. You'll need one flow entry *per connection* for forward traffic, but only one entry *per server* for return traffic.
- The switch port numbers are predictable; h1 resides on port 1, h2 on port 2, and h3 on port 3.

Regarding determining load:

Different workloads run afoul of different bottlenecks. You can use whatever metric you want for load, as long as it's sensible and realistic. For example, it you want to count the number of connections, try to eliminate those which are closing or taking advantage of HTTP keep-alive.

- You can use information provided by mod_status, netstat and just about any tool available in Linux.