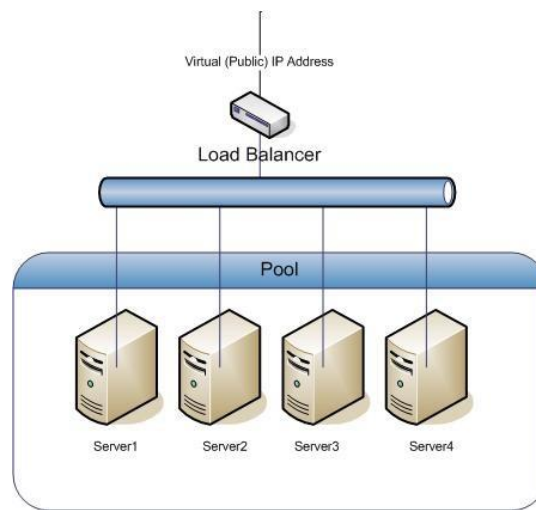


**Project Report (CS 6301.501)**

**SOFTWARE DEFINED NETWORKS**

**Topic:**

**ADVANCED LOAD BALANCER IN SDN**



**Team Members:**

**Divya Ammana**

**Piyush Supe**

**Raghav Mathur**

**Ravi Tej Rai**

## **Aim:**

To balance the network traffic over the SDN by dynamically assigning the requests on the server which has the lowest load or has the highest server capacity. We choose from (Weighted Round Robin, Random) techniques and take into account Statefulness and Statelessness of the architecture.

## **Types of Load – Balancing techniques Used**

### **Weighted Round Robin:**

The server with the highest capacity will be given the highest weight. For example, if Server 1 has 3 times the capacity of Server 2, then we assign Weight 3 to server 1 and weight 1 to server 2. This, means that the first 3 requests are sent to server 1 and the 4th request is sent to server. This continues cyclically for all requests.

### **Random:**

This technique uses a random number generator to determine a server that will handle the load. When load balancer receives large no. of requests, it will distribute the requests randomly among the servers so that the load is distributed equally among all the servers. it is suitable when the servers have similar configurations.

## **Types of States Used**

### **State-full ness and Stateless ness:**

In the case of state-full-ness the load balancer remembers the state when a client sends a request and assigns the future requests from that client to the same server. In the case of statelessness, no such state is remembered by the load balancer. Instead, for each new request, the client needs to log in again and then the request is forwarded to some server depending on the load balancing technique.

## **Implementation:**

**Note:** We are running POX controller with the required python script of the load balancer on the in one mininet instance. While all the topology and other commands are executed on another mininet instance. Both of them are terminals of the same virtual machine.

### **State-less – weighted Round Robin:**

Stateless means no Client- Server information is stored by the load balancer. Server 1 has weight 3 (assigned in python script), server 2 has weight 2 and server 1 has weight 1. This implements the **weighted round robin part**. The first three requests by ANY client are forwarded to the server 1. The next 2 are forwarded to server 2 and the 6th is forwarded to server 3 and this continues cyclically for all future requests. Now, to spot the difference between stateless and state-full, you need to observe the value of **nw\_src**. That represents the server. So, we consequently use h1 to send requests to the load balancer (see the terminal on the right). The load balancer will assign the first 3 requests to server 1, the next 2 requests are assigned to server 2 and the 6th request is assigned to server 1. This implies it is **Stateless** because if it was state-full all requests by host 1 would be sent to the same server.

The image shows two terminal windows side-by-side. The left window displays network configuration logs for 'libopenflow\_01', including fields like 'nw\_dst', 'nw\_src', 'tp\_src', and 'tp\_dst'. The right window shows a directory listing for '/<h2>' and a list of files with their permissions and sizes. A text box on the right side of the image states: 'It assigns the first 3 requests to server 2, followed by 2 requests to server 3 and 1 request to server 4'.

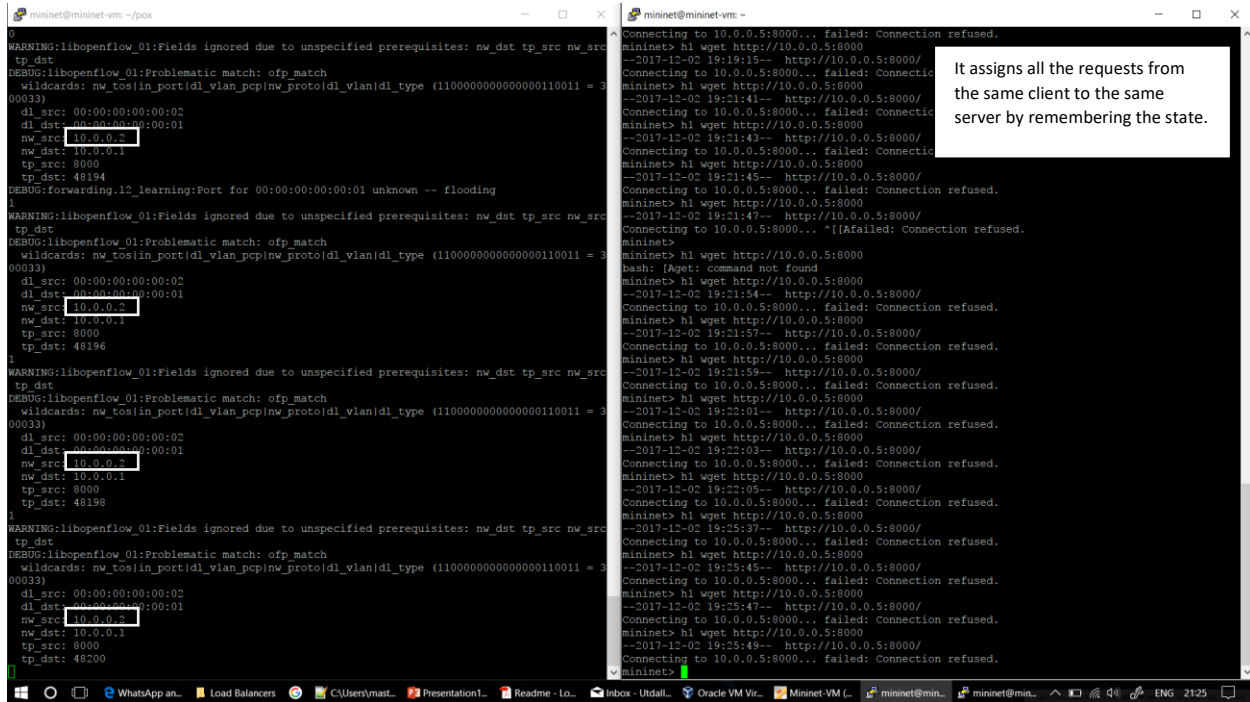
## State-less – Random:

Stateless means no Client- Server information is stored by the load balancer. So, we consequently use h1 to send requests to the load balancer (see the terminal on the right). The load balancer will assign the the requests to any of the 3 servers-h2, h3, h4 randomly and does not remember the state of the request. This implies it is **Stateless** because if it was state-full all requests by host 1 would be sent to the same server.

The image shows two terminal windows side-by-side. The left window displays network configuration logs for 'libopenflow\_01', including fields like 'nw\_dst', 'nw\_src', 'tp\_src', and 'tp\_dst'. The right window shows a directory listing for '/<h2>' and a list of files with their permissions and sizes. A text box on the right side of the image states: 'It randomly assigns all the requests to any server.'

## State-Full Weighted Round Robin:

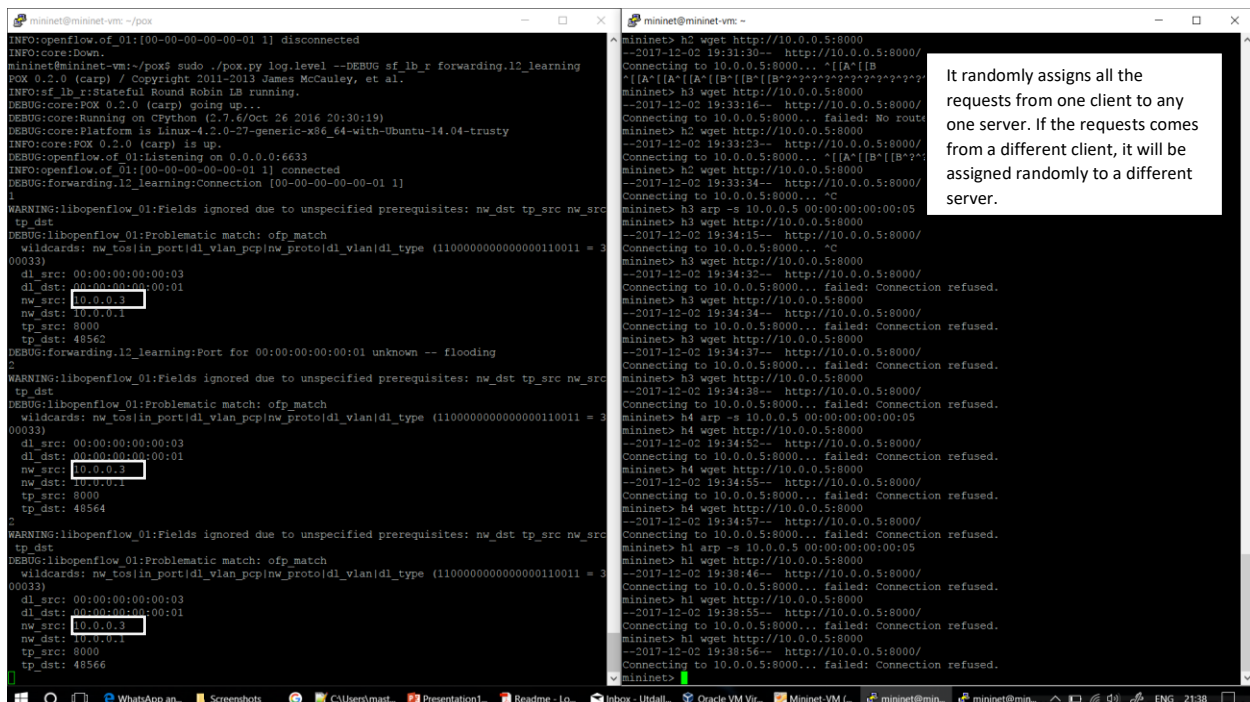
In this case the load balancer remembers which client is assigned to which server and any consequent request by this client is forwarded to the same server. Only when new clients send requests to the load balancer, it is assigned to a new server which has the next largest weight. Host 1 (h1) is sending the request to the load balancer. Since, host 2 (server) has the highest weight, this request is assigned to host 2. Any subsequent requests from host 1 are assigned to same server (host 2) by the load balancer.



The screenshot shows two terminal windows. The left window displays network traffic logs with fields like 'dl\_src', 'dl\_dst', 'nw\_src', 'nw\_dst', 'tp\_src', and 'tp\_dst'. The right window shows the load balancer's internal state and request handling, including connection attempts and failures. A text box on the right states: 'It assigns all the requests from the same client to the same server by remembering the state.'

## State-full – Random:

In this case the load balancer remembers which client is assigned to which server and any consequent request by this client is forwarded to the same server. The selection of servers is made randomly. Only when new clients send requests to the load balancer, it is assigned to some other server which is chosen randomly.



The screenshot shows two terminal windows. The left window displays network traffic logs with fields like 'dl\_src', 'dl\_dst', 'nw\_src', 'nw\_dst', 'tp\_src', and 'tp\_dst'. The right window shows the load balancer's internal state and request handling, including connection attempts and failures. A text box on the right states: 'It randomly assigns all the requests from one client to any one server. If the requests comes from a different client, it will be assigned randomly to a different server.'