CS 6250 - Computer Networks - Project Group 6

Team: Aarthi Vishwanathan (903390832), Anirudh Sarma (903298215), Archit Bansal (903396126)
Havish Chennamraj (903201642), Shreya Rajkumar (903390392)

Project Overview:

The goal of the project is to build an **SDN Load Balancer** and **Firewall** module.

Related Work:

We primarily surveyed literature regarding load balancing algorithms and firewalls. We drew inferences from the papers as to a possible implementation of a few in our project.
Among the several types of load balancing algorithms, they can be broadly classified into static and dynamic.
A few are described below:

- **ROUND-ROBIN:** In this, each servers receive the request from clients in circular manner. The requests are allocated to various live servers on round robin base.[1]
- **WEIGHTED ROUND-ROBIN:** In this, each server receive the request from the client based on criteria that are fixed by the site administrator. In other world A Static weight is assigned to each server in Weighted Round Robin (WRR) policy. We usually specify weights in proportion to actual capacities. So, for example, if Server 1's capacity is 5 times more than Server 2's, then we can assign a weight of 5 to server 1 and weight of 1 to server 2.[1]
- **RANDOM STRATEGY**: From a list of live servers, the Load Balancer will randomly choose a server for sending request. This policy has large overheads.[1]
- **HASH BASED.** The algorithm works by first calculating the hash value of traffic flow using IP address of source and destination, port number of source and destination and URL. The request is then forwarded to the server with highest hash value. If any other request comes with same hash value, it will be forwarded to same server. [2]
- **GLOBAL FIRST FIT**(GFF): After receiving a new flow request, the scheduler searches linearly all the available paths in order to find the one which can accommodate the bandwidth requirement of this new flow. The flow is greedily assigned to the first path which is fulfilling the requirement. Global First Fit does not distribute flows evenly across all paths.[3]
- **FLOW BASED LOAD BALANCING:**
    - Classify the incoming request into mice or elephant flows
    - The server selection module will select the server with the smallest elephant flow counter value or mice flow counter value depending on whether the new flow is elephant or mice, respectively.
    - Collect port statistics and flow statistics of switches. The statistics help in the selection of Server/Path based on a weighted heuristic. [3]

Firewalls have been implemented using simple rules based on MAC, IP addresses, protocol etc[4]

Progress Details:

To start with the implementation of the Load Balancer, we first setup **Mininet** and **Floodlight** in our systems.

For any SDN based study, the most important thing is the choice of the controller. After thorough discussion on the various SDN controllers available and weighing their pros and cons, the team decided to choose the Floodlight controller. Two of the main reasons for this choice include:

a. The fact that the Floodlight website had several coding examples along with directions on how to build the product. This was helpful to members who were new to this domain.
b. REST APIs are included to simplify application interfaces to the product.

After a good understanding of the working of the entire setup, we decided to implement a very simple load balancing module to start with. For this, we implemented a static rule based load balancer which works as follows:

The module uses Floodlight REST APIs to push static load balancing rules information to Floodlight. The rule contains a static (hard coded) destination ip address and the link to forward the traffic on. This way we were able to achieve two different packets with different destination ip being forwarded to different interfaces as per the rules pushed by us.
Once the rules were loaded into the controller, we verified the correct functioning of the module through packet tracing using Wireshark.

Getting the above module working end to end essentially means that we have all the building blocks ready to quickly implement a dynamic load balancing and firewall module.
Hence, we are now progressing towards implementing a load based approach wherein a path is selected based on the least load on the path towards the server, irrespective of the type of flow.

Two of the team members have finished exploring the implementation details of the firewall and will soon be writing a module for the same.

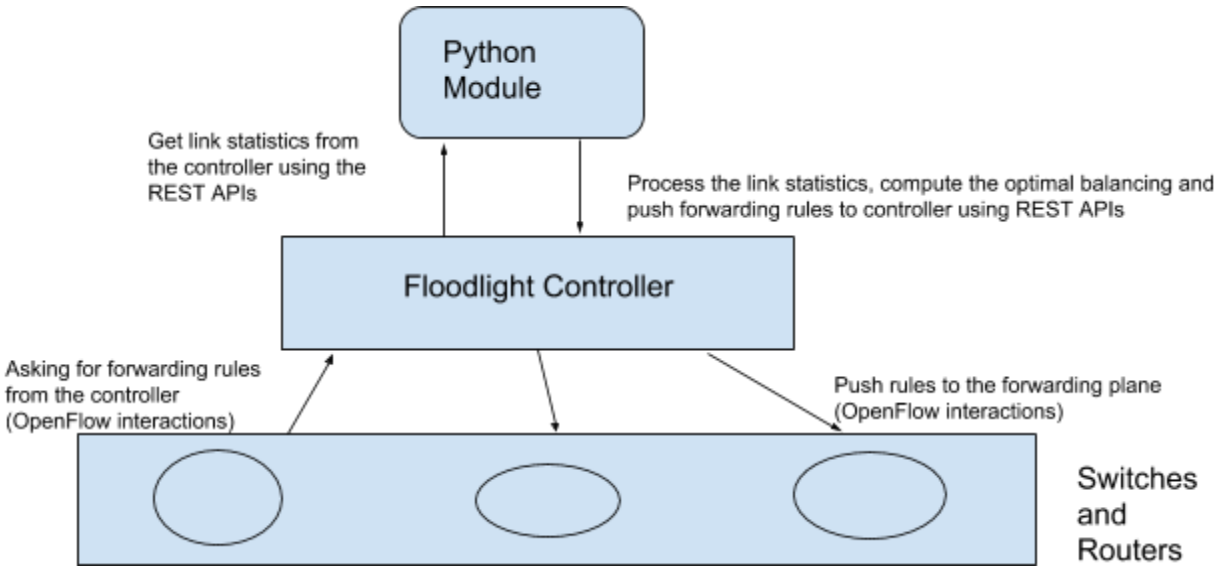
What has been done:

1. Mininet Setup and hands on.
2. Exploring controllers.
3. Setup Floodlight.
4. Full understanding of OpenFlow protocol.
5. Hands on with full setup of mininet, ovs switches and Floodlight.
6. Basic module implementation to add flow rules in controller using REST APIs.

7. End to end working of a static rule balancing the traffic among different links.
8. Exploration and hands on with the REST APIs exposed by Floodlight.
9. Identifying the REST APIs needed and useful to implement the final version of the load balancer and firewall.
10. Exploration of scope of Firewalls.

Methodology of the experiments with initial module:

1. Start up all the components required for running, implementing features and testing in SDN environment. This include mininet, floodlight controller and the python module.
2. Use Mininet to create custom topologies, generate traffic with different parameters (packet rate etc.), give different parameters (bandwidth characteristics) to different links and use wireshark to see the behaviour in each case.
3. Load rules in Floodlight controller through our rudimentary module, observe and validate the change in network behaviour through wireshark.



Interactions among the components in the SDN load balancer

What needs to be done:

1. Refine the rudimentary module to dynamically balance the load based on the statistics of each link available.
2. Refine the interface and add error handling.
3. Implement firewall module that is capable of pushing firewall rules into the controller and subsequently into the switches.
4. Document the end to end methodology and outcomes.

Change in scope:

Instead of the initially proposed web server to push load balancing rules into the network, we are now writing a self balancing module that runs at regular intervals and dynamically pushes rules in the network such that each link is equally utilised.

References:

[1] Weighted Round-Robin Load Balancing Using Software Defined Networking - Sabiya,Singh 2016
[2] Load Balancing in Software Defined Network - Kumari, Thakur
[3] Server and Network Load Balancing in SDN Content Delivery datacenter network
[4] Building L2-L4 Firewall using Software Defined Networking - Amandeep, Vikramjit