A Framework for Performance Analysis of Various Load Balancing Techniques in a Software-Defined Networking Environment

Patrick Von Angelo V. Atienza and William Emmanuel S. Yu

Abstract Load balancer is an essential part of a computer network. Its primary purpose is to distribute incoming traffic across multiple target servers. There are numerous load balancing techniques and each of them excels on specific network topology and server capability. However, due to vendor dependency, implementing a quintessential load balancer requires additional hardware cost and knowledge in vendor-specific configurations. Using software-defined networking (SDN) approach, testing of various load balancing techniques becomes easier and cheaper than traditional hardware-based approach. Despite the promising advantages of SDN, the novel approach is still unstable. Hence, in this experiment, performances of five different load balancing techniques—namely, random, round-robin (RR), weighted round-robin (WRR), least-connections (LC), and weighted least-connections (WLC)—were tested. The experiment was done on a single-switch topology. Mininet and POX controller were used to setup the network environment. The load balancers were also tested in two types of network conditions: with and without TCP SYN floods. After several iPerf tests, results in both network conditions indicated that RR and LC load balancers were both more than twice as fast as the one without load balancing implementation and moderately faster than random load balancer. LC and WLC were slightly faster than RR and WRR without SYN floods while RR and WRR were slightly faster with SYN floods. Future works, like testing the framework on other types of network topologies or low-level load balancing techniques, could strengthen the substantiation of stability of using SDN approach.

Keywords Software-defined networking · Load balancing · Mininet
POX controller · Openflow

P. V. A. V. Atienza (✉) · W. E. S. Yu
Ateneo de Manila University, Loyola Heights, Quezon City, Philippines
e-mail: patrick.atienza@obf.ateneo.edu

W. E. S. Yu
e-mail: wyu@ateneo.edu

K. J. Kim and N. Baek (eds.), *Information Science and Applications 2018*,
Lecture Notes in Electrical Engineering 514,
https://doi.org/10.1007/978-981-13-1056-0_7

1 Introduction

The main goal of software-defined networking (SDN) is to separate the control layer from the infrastructure layer to make the control layer programmable by end-users. With this emerging architecture, the network can be manipulated programmatically without touching the physical devices [1]. It also offers flexible, fast and cost-effective solution to continuously changing business requirements [1]. In traditional networking, it is impossible to split the infrastructure layer from the control layer. Therefore, end-users had to rely on the vendor for software network configurations and additional features. SDN tries to change that dependence in networking to an open networking system [2]. In SDN, a centralized controller manages the network flows that passes through network switches. SDN is developed to handle large networks like WANs, cloud computing networks and virtual networks. With the growth in today's network, data loss and degradation are highly susceptible; thus, an efficient algorithm that can handle large amount of load is necessary [2].

Load balancing is a method to distribute workload across multiple servers or other resources to achieve maximized throughput, minimized latency, and overload avoidance [3]. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. Load balancing is one of the initial steps of Quality of Service (QoS) networking. Depending on the specifications of each server, the load balancer can direct the flow of each request based on the type of request it receives. Studies, like in the research of Chato and Yu, successfully split network flows of HTTP and media streaming requests in an SDN network using various QoS mechanisms [4]. Load balancers can also be used as a first line of defense of DDoS attacks. It can reroute the network flow of all of the suspected packets to a single node or drop the packet completely. Guevara et al. successfully implemented the detection and dropping of suspected packet using an intrusion detection and prediction system over a software-defined network [5]. Load balancers are very handy with SDN because the network flow of each network switch can be guided programmatically by the control layer [3, 6]. In this research, we will compare five basic load balancing techniques, namely, round-robin, weighted round-robin, random, least-connections, and weighted least-connections. This research seeks to answer the following research questions:

1. What is the advantage of implementing a load balancer in the controller pane compare to forwarding the network flow to a single server?
2. Will each tested load balancing method can handle its stability in TCP SYN flood attacks? Which of the load balancing methods will perform the best?
3. Will all the load balancers perform well in terms of accuracy and throughput? Which of the load balancing methods will perform the best?

2 Theoretical Background

2.1 *Load Balancing Techniques*

Load balancers can be split up in two types: static load balancers and dynamic load balancers. In static load balancing, processes do not depend on the current state of the network and processes are assigned prior to the execution of the network. On the other hand, in dynamic load balancing, processes change from time to time and the system needs to be recalibrated in order to distribute load equally through servers. Below are the most common static and dynamic load balancing techniques:

Round-robin Load Balancing. RR is a static load balancing technique and it is one of the simplest methods for network flow distribution. Going down the list of servers in the group, the round-robin load balancer forwards a client request to each server turn by turn. When it reaches the end of the list, the load balancer returns back to the initial server and do the method again [3, 6]. The method of selecting a server can operate with the worst-case time complexity of O(1) since the details of the last selected server can be stored. **Weighted Round-robin Load Balancing.** In WRR, a weight is assigned to each server depending on its network capability, its traffic-handling capacity, or its power of processing the received data [8]. The higher the weight, the larger the proportion of client requests the server receives. WRR is also a static load balancing technique. This is useful for servers that have different specifications [3, 6]. This technique can also achieve the same worst-case time complexity as the ones in RR. **Least-Connections Load Balancing.** When a virtual server is configured to use LC, the load balancer selects the server with the fewest active connections. LC is a dynamic load balancing technique and one of the default methods in load balancing, because, in most circumstances, it provides the best performance [3]. The method of selecting a server has a worst-case time complexity of O(n) with n as the number of servers. The reason is that the load balancers should check all the active connections of each server. **Weighted Least-Connections Load Balancing.** WLC is almost the same as the non-weighted one with an exception all servers are being weighted depending on their network handling capabilities or their own performances of processing received packets. This technique can also achieve the same worst-case time complexity as the ones in LC.

2.2 *OpenFlow, Mininet, and POX Controller*

OpenFlow is one of the early standards of SDN. As defined in the paper of the McKeown et al., the main feature of OpenFlow is to have a full control of all data packets roaming around the network [7]. The movements of the data packets, or "flows" as commonly referred to it, are controlled through user-defined rules and protocols. The details of each flow entry can be recorded in a flow table which can be also controlled programmatically [7, 8]. Each flow entry has three fields: (1) the

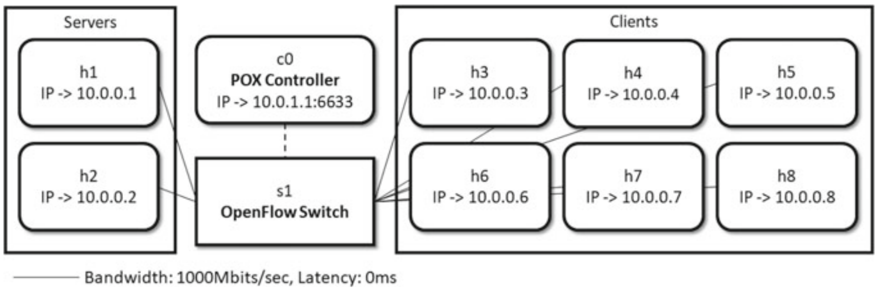

Fig. 1 The network topology to be set up using the Mininet network emulator

packet header that defines the flow or the "rule", (2) the action that the packets are process or the "action", and (3) the flow and port statistics or the "stats". An action can be also classified into four basic types: (1) forwarding packet to ports, (2) forwarding and encapsulation of packets to controllers, (3) dropping of packets, and (4) forwarding packet to normal processing pipeline [7, 8].

Mininet is a network emulator that creates a virtual network of Layer 2 and Layer 3 switches, controllers, and hosts. The switches that offered by Mininet are Open vSwitches which support OpenFlow. Mininet is often used in research and development, prototyping, testing and debugging, and other tasks that needs an experimental network simulation. It runs on Unix/Linux environment [9].

POX controller is one of the most common frameworks for simulating SDN controllers. It is patterned from the NOX controller and it is written in Python. The main advantage of POX is that it is easy to use and does not require a steep learning curve [10, 11]. The disadvantage is that it is slow compare to other controllers like OpenDayLight, and Floodlight. That is why POX controllers are often used for educational purposes [10].

3 Methodology

3.1 Network Topology

The single-switch topology consisted of a single Open vSwitch (S1) connected to eight hosts (h1, h2, h3, h4, h5, h6, h7, h8) with two hosts (h1, h2) as servers and six hosts (h3, h4, h5, h6, h7, h8) as clients (Fig. 1). A switch was controlled by a remote POX controller. Each data link had a bandwidth of approximately 1000 megabits per second and 0 ms latency. An additional client host added in the network topology for the latter part of the experiment which tested the performance of the network during TCP SYN floods. The additional host acted as a persistent bot host.

3.2 Mininet and POX Controller Configurations

The network topology can be created programmatically in Python. However, Mininet has already terminal line commands on creating various types of networks based on network topologies such as single-switch topology.

```
sudo mn --topo single,8 --controller = remote,port = 6633 --link tc,bw = 1000,delay = 0 ms
```

The POX controller has an event called `PacketIn`. It triggers every time a packet is received by the controller. The packet can be identify as TCP if it returns a value in `packet.find('tcp')` command. If the source IP address of the TCP packet is a client IP address, the controller performs the load balancing method. On the other hand, if the source IP is a server IP address, then the controller forwards the TCP packet with the ACK response to the destination client. As explained by Peña and Yu, flow entries could be installed and modified by sending a `ofp_flow_mode()` message that matched the attributes of the packet [8].

As for the random load balancer, the controller picked a random IP address of a live server. The POX controller already had a configuration of random balancer which could be used anytime. As for the RR load balancer, the controller picked a live server with the lowest IP address in the array of live servers and saved the index of the server in the memory. Whenever a packet from a new TCP connection had been received, the controller would pick the succeeding live server and its IP address would be also saved. The process repeated after the controller picked the live server with the highest IP address. As for the LC load balancer, `FlowStatsReceived` event was used to determine active flows in the flow table. Each live server had a number of connections that could be incremented if it had the least value. All TCP connections established by client hosts had unique source port numbers. The controller saved that port number, with selected live server's IP address, to the memory. Using a polling function, the controller checked each port if it still existed in the flow stats. If the port number did not exist in the flow stats, the port number would be deleted in the memory and the corresponding live server's connections will be decremented. As for the WRR, the index will be set a float value and it will be incremented by 1 divided by the weight of previously picked server. As a basic rule of programming languages, if a float value is parsed to an integer, the result would be the floor value of the float variable. By these, using the integer-parsed value of the index would get the corresponding live server that can be picked by the controller. As for the WLC load balancer, the number of connections of each live server will be also set as a float value. Each connection that will be added to a live server must incremented to 1 divided the weight of the chosen live server.

4 Results

The Mininet-based network was tested on five load balancing algorithms. Each host transferred 1000 MB (or approximately 8388.608 megabits) of data to the server and this was done in iPerf 2.0.5 As for the weighted load balancers, h1 had a weight of 2 and host h2 would have a weight of 1; however, the host servers still retained the same specifications as the ones with non-weighted load balancers. Each load balancer was tested 10 times and the result of each test was averaged.

As shown in Table 1, The network with no implemented load balancer only uses an average of 87.53% of the total bandwidth. In contrast to the networks with non-weighted load balancers, the throughput of the non-load balanced network is extremely lower. The LC load balancer is more efficient than both RR and random load balancers with throughputs 1.53% and 11.25% higher respectively. Even in weighted load balancers, WLC was 1.79% faster and had a 0.85% higher throughput than WRR load balancer. LC and WLC were faster than RR and WRR in this scenario since the controller could pinpoint the best server for every initiated connection.

As for the part where TCP SYN floods were included, an additional host was added to continuously send 6 parallel TCP requests to the network. As shown in Table 2, the network without an implemented load balancer was significantly slower because all of the load of the TCP flood attacks was carried by only one host. The non-load balanced network had the least network throughput as it only used 39.85% of the total bandwidth. The RR load balancer had the fastest transfer time amongst all load balancers. RR and WRR were slightly better than the LC and WLC as they finished the whole transfer process 3% faster for non-weighted and 4.5% faster for weighted. One of the main reasons why LC and WLC were slower than RR and WRR in this scenario was because a single controller handled the decision making of all initiated connections including those that initiated by bot host. Since RR and WRR had better worst-cast time complexity than LC and WLC, the overhead of

Table 1 iPerf results of all load balancing techniques without TCP SYN flood

Load balancing method	Number of connections	Transfer time (s)	Transfer rate (Mbits/s)	% throughput
None	h1: 6, h2: –	h1: 57.5, h2: –	h1: 875.33, h2: –	h1: 87.53%, h2: –
Random	h1: 3, h2: 3	h1: 26.8, h2: 31.4	h1: 939.02. h2: 801.46	h1: 93.90%, h2: 80.15%
RR	h1: 3, h2: 3	h1: 25.9, h2: 26.9	h1: 971.65, h2: 935.53	h1: 97.17%, h2: 93.55%
LC	h1: 3, h2: 3	h1: 25.6, h2: 26.4	h1: 983.04, h2: 953.25	h1: 98.30%, h2: 95.33%
WRR	h1: 4, h2: 2	h1: 39.7, h2: 17.6	h1: 845.20, h2: 953.25	h1: 84.52%, h2: 95.32%
WLC	h1: 4, h2: 2	h1: 39.0, h2: 17.6	h1: 860.37, h2: 953.25	h1: 86.04%, h2: 95.32%

Table 2 iPerf results of all load balancing techniques with TCP SYN flood

Load balancing method	Number of connections	Transfer time (s)	Transfer rate (Mbits/s)	% throughput
None	h1: 6, h2: –	h1:126.3, h2: –	h1: 398.51, h2: –	h1: 39.85%, h2: –
Random	h1: 2.5, h2: 3.5	h1: 34.6, h2: 58.7	h1: 484.89, h2: 571.63	h1: 48.49%, h2: 57.16%
RR	h1: 3, h2: 3	h1: 45.4, h2: 53.3	h1: 554.31, h2: 472.15	h1: 55.43%, h2: 47.22%
LC	h1: 3, h2: 3	h1: 40.2, h2: 54.9	h1: 626.02, h2: 458.39	h1: 62.60%, h2: 45.84%
WRR	h1: 4, h2: 2	h1: 57.8, h2: 26.2	h1: 580.53, h2: 640.35	h1: 58.05%, h2: 64.04%
WLC	h1: 4, h2: 2	h1: 60.4, h2: 27.8	h1: 555.54, h2: 603.50	h1: 55.55%, h2: 60.35%

establishing each connection was less. It was evident that the penalty of the overhead was more prevalent than the benefit of the precise decision making of LC and WLC.

5 Conclusion and Future Works

The load balancers are successfully implemented in the POX controller. As the results show, implementing a load balancer within the control layer exceedingly increase the throughput of the network flow and decrease the transfer time by a hugely large amount. The results also show that the LC and RR load balancers are similarly efficient in terms of their throughput. However, a straightforward LC approach can be inefficient in terms of TCP requests because of the TIME_WAIT state of the TCP protocol that usually lasts around 2 min or less depending on its configuration. This can be problematic in a busy network and may possibly lead to an unstable load balancer.

This experiment is made to show the capability of an SDN controller to balance the requests of all clients. Most of the load balancing techniques are difficult to implement in a lower level network switched if SDN is not applied on the network topology. Future works like testing the performance of load balancers in other types of network topology or other types of low-level load balancing techniques, like Source IP Hashing and Least Packets, could strengthen the substantiation of the stability of using the software-defined networking approach. Implementing a testbed with a distributed controller system instead of a single controller could also avoid the bottleneck of the network.

References

1. Kreutz D, Ramos F, Verissimo P, Rothenburg C, Azodolmolky S, Uhlig S (2015) Software-defined networking: a comprehensive survey. Proc IEEE 103(1):14–76
2. Xia W, Wen Y, Foh CH, Niyato D, Xie H (2015) A survey of software-defined networking. IEEE Commun Surv Tutor 17(1):27–51
3. Bhandarkar S, Khan KA (2015) Load balancing in software-defined network (SDN) based on traffic volume. Adv Comput Sci Inf Technol (ACSIT) 2(7):72–76
4. Chato O, Yu W (2016) An exploration of various quality of service mechanisms in an OpenFlow and software defined networking environment in terms of latency and performance. In: 3rd international proceedings on information science and security (ICISS). IEEE, pp 1–7
5. Guevara AG, Domingo MA, Yu W (2017) Enhancing intrusion detection and prevention systems using software defined networking in a distributed topology. In: 17th proceedings on philippine computing science congress. CSP, Quezon City, Philippines, pp 219–228
6. Kaur S, Kumar K, Sing J, Ghumman N (2015) Round-robin based load balancing in software defined networking. In: 2nd international proceedings on computing for sustainable global development (INDIACom), IEEE, pp 2136–2139
7. OpenFlow Switch Specification. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. Accessed 17 Jan 2018
8. Peña JG, Yu W (2014) Development of a distributed firewall using software defined networking technology. In: 4th International Proceedings on Information Science and Technology (ICIST), IEEE, pp 449–452
9. Lantz B, Handigol N, Heller B, Jeyakumar V (2018) Introduction to Mininet. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. Accessed 17 Jan 2018
10. McCauley M (2018) POX Wiki. https://openflow.stanford.edu/display/ONL/POX+Wiki. Accessed 17 Jan 2018
11. Prete L, Shinoda A, Schweitzer C, de Oliveira R (2014) Simulation in an SDN network scenario using the POX controller. In: Proceedings on communications and computing (COLCOM). IEEE, pp 1–6