International Journal of Engineering & Technology, 7 (3) (2018) 1854-1857

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i3.14017
Research paper

Load balancing strategies in software defined networks

Gourav Shrivastava [1] *, Praveen Kaushik [1], R. K. Pateriya [1]

[1] Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, Bhopal, India
**Corresponding author E-mail: gashr83@gmail.com*

Abstract

In the past few years, network requirements have been changing frequently as the amount of data traffic increasing exponentially so it is difficult to utilize the full capacity of network resources. Software Defined Networking (SDN) is emerging as a new networking technology which decouples the control plane from the data plane in the network devices. Separation of control and data plane allows a network administrator a better control over network management and also enables new development through network programmability. Presently Open-Flow is the most popular SDN protocol which provides communication between network devices and controller. In this paper, the Round Robin algorithm is compared with the Dynamic load balancing algorithm using the OpenFlow protocol in SDN under varying load conditions of TCP and UDP traffic. Experimental analysis shows that the dynamic load balancing strategy works better than the Round Robin load balancing.

Keywords: *Load Balancing; Open Flow; Software-Defined Networking Semicolon.*

1. Introduction

In Conventional network architecture, the network devices are vertically integrated means the hardware and software is manufacturer specific which can't be customized [1] new software or policies may not be installed because of incompatible hardware, or the currently available software couldn't leverage all the hardware capabilities. Also, there is no arrangement of finding the global view of the network. Today's network devices communicate with each other and are not able to select a path from a global view. These issues are some of the challenges that have motivated the researchers to move forward with some new ideas in networking.
Software Defined Networking is a consequence of such necessities. SDN is an example of programmable networks. The basic idea behind SDN is the decoupling of the data plane from the control plane. Control Plane decides what is to be done and tell the data plane to implement the decision. Control plane has the ability to control and forwarding behavior like computing routes, tracking topology changes, install forwarding rules etc. On the other hand, data plane only forwards the traffic based on rules as dictated by control plane logic. The centralized control plane called the controller will control data planes and can be implemented completely in software. This architecture provides a global view of a network the controller is able to see the status of all routes and switches for quickly deciding the best route.
The rest of the paper is organized as follows. Section II gives an overview of the architecture of SDN and OpenFlow switch. Section III presents some related work done in this area. Section IV describes the load balancing strategies. Section V explains the evaluation setup and provides the results. Finally, section VI concludes the paper.

2. Architecture of SDN

ONF (Open Networking Foundation) [2] suggested a reference model for SDN, as illustrated in Fig 1. The SDN reference model has three layers an infrastructure layer, a control layer, and an application layer. The infrastructure layer is the lower level layer, which consists of networking devices (e.g., switches, routers etc.) in the data plane. These networking devices are responsible for collecting network status, storing them temporarily and sending them to controllers by an open API called OpenFlow. The network status may hold information such as network topology, traffic statistics, and network usage. These devices are also responsible for processing packets based on rules provided by a controller. The control layer works between the application layer and the infrastructure layer, via its two interfaces. With the infrastructure layer (i.e., the south-bound interface), it specifies functions for controllers to access functions provided by switching devices, for example, reporting of network status and importing packet forwarding rules [1] and with the application layer (i.e., the north-bound interface), it provides service access points in various forms, for example, an Application Programming Interface (API). SDN applications are able to get network status reported from network devices through this API, enables the system to take decisions based on this information, and carry out these decisions by setting packet-forwarding rules to networking devices using this API [1]. The top-level layer is the application layer, which contains SDN applications designed to fulfill user requirements. Examples of SDN application [1] are dynamic access control, seamless mobility and migration, load balancing, and network virtualization. The SDN Controller is taking care of maintaining the network flow rules and gives instructions to the underlying infrastructure on how traffic should be handled. Software-defined network can be used to manage large networks like data center. Performance and efficiency of a network degrade due to the large traffic load on the links. So, there is a need of an efficient routing algorithm which

can handle or manage high volume of traffic by balancing load among all the links available.

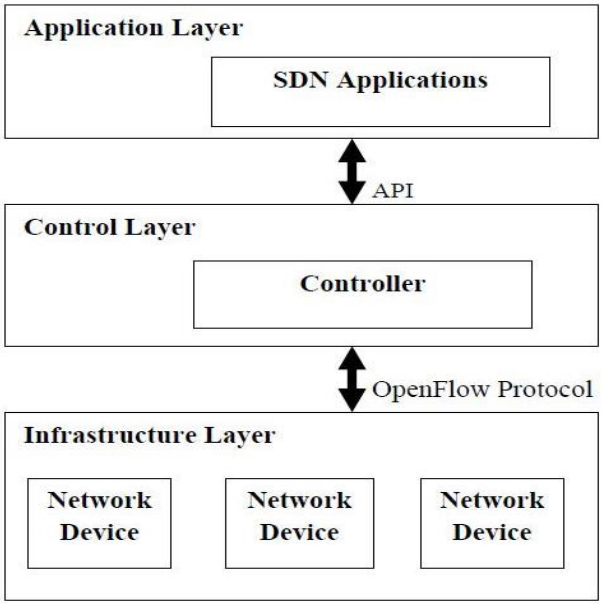

Fig. 1: SDN Reference Model.

2.1. Open flow switch

OpenFlow Switches gives an open, programmable, virtualized platform to enable deployment of new protocols, while network administrators can ensures that the device is well supported [1]. An OpenFlow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller as shown in Fig 2. The switch communicates with the controller and this communication is managed via the OpenFlow protocol.

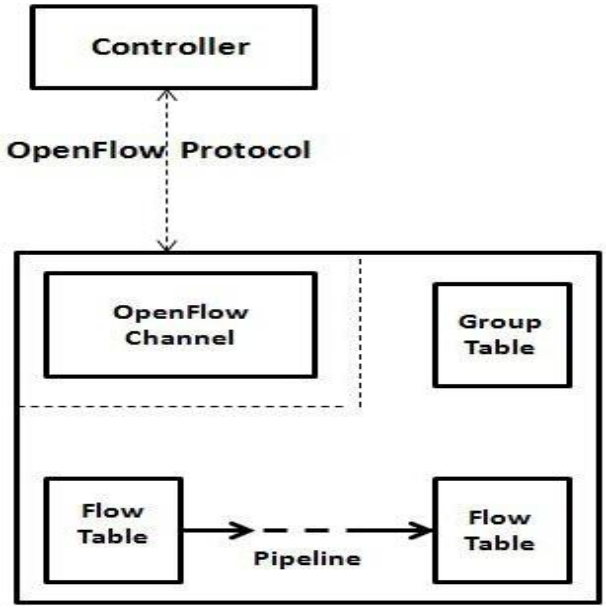

Fig. 2: Open Flow Switch.

3. Related work

Load Balancing is a technique used to distribute large number of requests across multiple paths. Load balancing increases network performance by properly using the available paths and helps in improving throughput and latency of the network. The SDN gives facility to design routing algorithms on top of SDN switches and enables to do load balancing accordingly. N. Handigol, et.al, [3] proposed a system, called Plug-n-Serve, and minimizes the response time by controlling the load on the network and the servers using customized flow routing but it has scalability limitations. Richard Wang et al. [4] propose a more scalable solution with algorithms that compute simple wildcard rules to reduce the load on the controller and automatically adjust to changes in load balancing policies. Hardeep Uppal et al. [5] implemented a load-balancer architecture based on the OpenFlow technology which reduces the cost and provides the flexibility. M. Koerner and O. Kao [6] proposed a load-balancing algorithm for handling multiple services (called LBMS) by using Software Defined Networking. It uses the FlowVisor, an SDN device to achieve network virtualization, to coordinate multiple controllers, each of which handles requests destined for different services. H. Long et.al [7] proposed a load-balancing algorithm, named LABERIO by considering the path and link utilization as a method to optimize the system throughput.

4. Load balancing strategies

To increase the throughput and minimize the latency of a network dynamic management of network resources is required. The load balancing in the network can be achieved by implementing balancing algorithms on switches to process packets. Floodlight [9] controller uses Round-Robin (RR) Algorithm for processing packet requests. There are multiple ports are available in the switches but in practice, some ports are least used while others are frequently used. This paper analyzes the round-robin scheduling algorithm and dynamic load balancing algorithm.

4.1. Round robin (RR) scheduling algorithm

RR Process packets received at switch ports in a rotating manner, execution i=(i+1) mod n in each time, and choose the packet sequence with a unique destination address for a time slice and iterates over them. The algorithm is least concerned about the traffic on the destined path, It causes the switch load imbalance. If there exist huge difference between the traffic generated then the algorithm causes waiting time and is not suitable for this type of scheduling policies, but when each of the packet sequence at ports generates the same amount of traffic then the algorithm suits best.

4.2. Dynamic load balancing algorithm

In this algorithm we first collect the information of hosts such as their IP, Switch to which they are connected, MAC Addresses, Port mapping, etc after collecting the information it obtains path information using Dijkstra algorithm thereby limiting the search to shortest paths between the hosts on which load balancing has to be performed in the fat-tree topology and find total link cost for all the paths available between any two hosts. Then the flows are created depending on the minimum transmission cost of the links at the given time as shown in. By finding the path using the Dijkstra algorithm which has minimum cost is selected and new flows are installed on each switch in the selected path. Information such as In-Port, Out-Port, Source IP, Destination IP, Source MAC, Destination MAC is given by the flows.

4.2.1. Algorithm to perform dynamic load balancing

```
Step 1: getting devices information by controller
While true:
request.put (statistics)
Get response (deviceinfo, "deviceInfo")
loadbalance ()
 getresponse (url,choice)
If (choice=="deviceinfo")
"Device information (data)
Else if (choice=="findswitchlinks")
Find switch links (jdata,switch)
Else if (choice == "linktx")
linktx (jdata,portkey)
```

Step 2: getting MAC addresses
Device information (data)
For i in data;
Device mac()
Step 3: get links of switches & adding links of ports between switches findswitchlinks (data,s)
Step 4: getting routes between source and destination hosts find switch routes ();
Routes (src,dst)
Step 5: compute link cost
linkTX (data, key)
Getting cost ()
Step 6: add new flow rules
Create flow rules ()
addflowrules ()
Step 7: perform load balancing
Loadbalance ()
Get response (linkURL, "find switch links")
Find switch route ()
Get link cost ()
Add flow ()
End.

5. Evaluation and results

5.1. Topology used

For the experimental purpose, the fat-tree topology (Figure. 4) is used which involves multiple paths among hosts so it can provide higher bandwidth as compared to a single-path tree with the same number of nodes. A fat-tree topology has an advantage that for any switch, the number of links going down to its siblings is equal to the number of links going up to its parent in the upper level. Network switches are connected to the Floodlight SDN controller. The communication between the open flow switches and the controller is made by the OpenFlow protocol in the form of OpenFlow message format. The controller is solely responsible for adding flow rules to open flow switches and further switches follow these flow rules for sending traffic between hosts.
The fat tree topology used for experimental analysis is built in a python script. When the script is executed in Mininet it builts the fat-tree topology consisting of 8 hosts and 10 open flow switches. Floodlight controller which is running remotely continuously sending LLDP packets and whenever it finds any OpenFlow switches connected it immediately made connections with the switches. Once the controller is connected to the topology a script written in python is executed to perform dynamic load balancing, which calculates the shortest paths using the Dijkstra algorithm. For experimental purpose a host H4 which has IP address 10.0.0.4 configured as a server and rest as a host. Now a host H1 which has IP address 10.0.0.1 starts communication with the server H4, Iperf testing is used to generate traffic it also gives throughput which is then used for analysis.

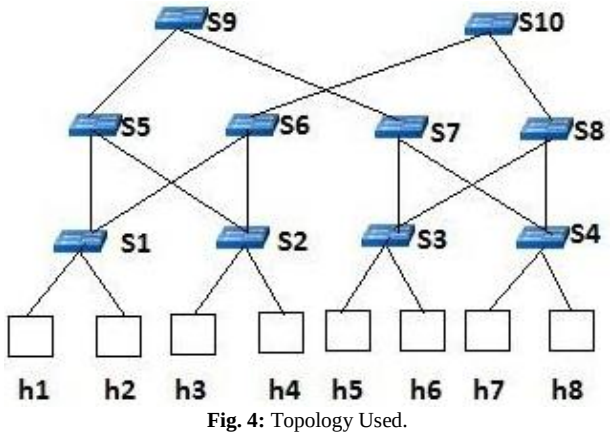

Fig. 4: Topology Used.

5.2. Simulation tool

Mininet [8] is a platform that allows rapid prototyping of large networks on a single computer. The Mininet supports Software Defined Network elements and gives facility to customize them. These elements include hosts, switches, controllers and links. The OpenFlow switches created by Mininet provide same functionality as in a hardware switch. Floodlight [9] is an OpenFlow controller for enterprise networks based on Java programming language and distributed under the Apache license. It is offered by Big Switch Networks that works with the OpenFlow protocol to manage traffic flows in a SDN environment. to better adapt to their changing needs and have better control over their networks.

5.3. Simulation setup

Load balancing strategies has been implemented using Floodlight-1.2 controller with Open-Flow protocol-version 1.3 on a system with Intel(R) Core(TM) i7-3770, 3.40 GHz, 4.00GB RAM and Ubuntu 14.04 OS. Mininet is used to create fat tree topology consisting of 10 OpenFlow switches and 8 hosts. Number of flows per second and number of responses given by the controller per second can be calculated by Iperf. Initially, test conducted between two hosts which are connected through different switches. To test the results first the topology is connected with floodlight using RR load balancer and then the dynamic load balancing algorithm are analyzed and performs load balancing between any two host in the fat-tree topology results shows that after applying the dynamic load balancing better throughput is achieved.

5.4. Comparison and analysis

Iperf, an open source network performance measuring tool is used to perform network throughput tests. Iperf can be used to calculate different metrics like bandwidth, jitter and datagram loss in the case of UDP packets. UDP is an unreliable protocol, datagrams are lost in RR is more than dynamic scheduling as the target bandwidth increases shown in table 1 and table 2 shows that the dynamic scheduling achieved better bandwidth and transferred more data than RR for TCP traffic also.

Table 1: Load Balancing Under Different Bandwidths for UDF Traffic

Interval (sec)	Target Bandwidth (Mbps)	Data transferred (MB)		Datagrams lost (%)		Jitter (ms)	
		RR	Dynamic	RR	Dynamic	RR	Dynamic
0-10	1	1.19	1.19	0	0	0.018	0.009
0-10	10	11.9	11.9	0	0	0.013	0.015
0-10	100	120	120	0	0	0.002	0
0-10	200	239	241	0	0	0.003	0.006
0-10	500	606	590	0.14	0	0.004	0.002
0-10	1000	884	938	0.048	0.003	0.001	0.001

Table 2: Load Balancing Performance for TCP Traffic

Interval (sec)	Data transferred (MB)		Bandwidth(Mbps)	
	RR	Dynamic	RR	Dynamic
0-120	209	248	14.9 Gbps	17.8 Gbps

6. Conclusion and future scope

In Software Defined Networks control is logically centralized sometimes the load on some links are very high while on some links it is very low thus capability of the network decreases. By dynamically balancing the loads on links by shifting the load on

best-calculated path reduces the congestion and information loss. This paper analyses the round robin algorithm used in floodlight controller and Dynamic load balancing algorithm. Experimental analysis shows that Dynamic load balancing gives better result than the round robin algorithm used in floodlight controller. In the future work the algorithm can be extended to work with the realistic network traffic.

References

[1] F. Hu, Q. Hao and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181-2206, Fourthquarter 2014.

[2] Open Networking Foundation (ONF),SDN Architecture, Issue 1, ONF TR-502 (2014).

[3] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-Serve: Load-balancing web traffic using Open Flow*". Demo at ACM SIGCOMM* (2009).

[4] Wang R, Butnariu D, Rexford J, "OpenFlow-based server load balancing gone wild", *In: Proceedings of the 11th USENIX conference on hot topics in management of internet, cloud, and enterprise networks and services*, pp 1212, (2011).

[5] Uppal H, Brandon D, "OpenFlow based load balancing", *In: Proceedings of CSE561: networking*. Project report. University of Washington (2010).

[6] M. Koerner and O. Kao. "Multiple Service Load-Balancing with Open Flow", *IEEE 13th International Conference on High Performance Switching and Routing* (2012).

[7] H. Long, Y. Shen, M. Guo and F. Tang, "LABERIO: Dynamic load-balanced Routing in OpenFlow-enabled Networks," *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, Barcelona, 2013, pp. 290-297.

[8] Bob Lantz, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks" *In Proceedings of the ninth ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*. ACM, New York, NY, USA, , Article 19 (2010).

[9] Floodlight Controller URL http://www.projectfloodlight.org/floodlight