Load Balancing Implementation in Software Defined Networks

Nikolay S. Prodanov[1], Kamelia S. Nikolova[1], and Dimitar K. Atamian[1]

Abstract – **Software defined networks (SDN) demonstrate a number of benefits compare to traditional telecommunication networks with respect to efficient utilization of network resources. In order to further improve the network performance a load balancing approach in SDN is proposed in this paper. POX controller and Mininet software are used for the experimental setup.**

Keywords – **software defined networks, load balancing approach, POX controller, Mininet.**

I. Introduction

Software defined networks (SDN) are innovative, rapidly evolving and promising technology. It is based on disassociating the forwarding process of network packets (data plane) from the routing process (control plane) [1]. This separation leads to some major advantages:

- The control plane is implemented in the SDN controller – thus control protocols and the accompanying complex solutions and algorithms are executed in one centralized location (SDN controller) with high computing capabilities. Thus allowing centrally controlling and managing the entire network.
- The data plane is implemented in SDN devices (commonly called SDN switches, or network elements) – in this way, it is possible to simplify the devices in the infrastructure plane and to use general purpose hardware without additional computational requirements, only with the availability of network interface cards.

SDN controller uses Application Programming Interfaces (APIs), to communicate with other controllers within the same administrative domain (eastbound interface) or among different administrative domains (westbound interface); in order to interpret and describe the SDN applications requirements (northbound interface) and to communicate and manage devices in the infrastructure plane (southbound interface) [1]. The main role in this innovative concept is concentrated into the functionality of the southbound interface, where the OpenFlow protocol [2] is the most used and dynamically developing solution [3].

Using the OpenFlow protocol, the SDN controller injects flow processing rules into SDN switches into so-called flow tables. One entry in the flow table is defined for each flow. Thus, depending on the SDN applications and their QoS requirements, different streams can be very finely defined. In order to achieve the desired QoS, one of the main tasks in the design of the control plane is to ensure high availability and subsequent minimization of the delay between SDN switches and SDN controller. As a result, load balancing is a critical solution that must be implemented effectively to manage inbound traffic and available resources in order to improve network performance. This functionality is implemented within the SDN controller.

The main aim of this publication is to develop the implementation of various techniques for load balancing in the SDN environment and to compare their performance.

Our contribution in this paper is:

- Creating Round Robin load balancing code;
- Creating Least connection load balancing code;
- Creating Dynamic feedback load balancing code;
- Creating host discovery code;
- Creating experimental setup for the comparative study using two scenarios using Mininet emulator.

The rest of the publication is organized as follows. In Section II, you can find different load balancing techniques. Section III describes the experimental setup and the implementation performed. Section IV presents the results obtained in tabular form. Concluding remarks are presented in Section V.

II. Related Work

Load balancing is a technique for improving the usability of available resources (the so called performance), which distributes the load based on a predefined policy depending on various application characteristics (flow, port, delay, etc.), depending on the available hardware network resources (memory, CPU, load, delay, etc.), depending on certain patterns of behaviour, etc. Classification and comparative study of the existing load balancing solutions are proposed in [4] as well as in the reference therein. In [4] they are classified into six categories as follows: static or policy-driven, random strategy based, dynamic or feedback driven, centralized, decentralized and clustering based. In the SDN environment, the OpenFlow protocol provides better network programmability, which can achieve much more flexible and efficient load balancing compared to traditional telecommunications networks. As a result, in addition to the well-known existing techniques and algorithms for load balancing [4], some new ones can be developed or adapted according the SDN architecture feature [5][6].

The following load balancing techniques are used in this work as the most general and universal: Round Robin [7], Least Connection [4], and Dynamic feedback [4][8]. The first two load balancing strategies are one of the most used by service

[1]The authors are with the Faculty of Telecommunications at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, E-mails: jitzerttok98@gmail.com , ksi@tu-sofia.bg and dka@tu-sofia.bg .

providers such as Amazon Elastic Load Balancing, Microsoft Azure, Rackspace, HP Cloud Load Balancing and GoGrid [4]. In this work, the selected load balancing methods are implemented in SDN environment.

Round Robin load balancing is a static strategy, where the load is distributed to the servers sequentially in a cycle without taking into account the environment characteristics such as number of server connections, response time, usability of computing power and many other factors. Thus, this method is easy to be implemented and is embedded in almost every load balancing device. Based on this method, various implementations have been proposed, by adding some new conditions in the algorithm [4][5].

Least connection balancing strategy aims to distribute the load based on the number of active connections on each of the servers. When a new request arrives at the balancing device, the server with the least number of connections is selected. This is a dynamic strategy, as it takes the environment into account when deciding which server to send the incoming request to [4].

Dynamic feedback load balancing is an algorithm that takes into account in real time environmental variables such as server load, service time, and line delays [4][8]. The aim is to unload servers that are busy with complex operations and to redirect incoming traffic to less loaded servers. To achieve such a distribution, a method to measure the environmental parameters must be defined firstly. Parameters such as the number of active connections to a server machine can be relatively easily retrieved from the cached information in the balancing device. Other parameters such as server load cannot be determined directly by the balancing device. Therefore, it is necessary to form feedback between the server machine and the balancing device. There are several ways to achieve this feedback. In most cases, this is done within the IP network, and protocols such as SSH (Secure SHell) can be used, through which the balancing device can access the server and download statistics on its resources used. Another method, again implemented through network connections, is to install a program (daemon) on the server machine to monitor the status of the machine and periodically report the current status of the machine to the balancing device via TCP connection or UDP packets. This method is also called agent-based, and the installed program is "agent". The advantage of agent-based feedback compared to the balancing device periodically downloads the necessary information, is that for the agent, the connection can only be one-way and, secondly the agent can declare the server inactive in case of partial failure or when the server is in maintenance mode. The disadvantage of the agent is that there may be some delay in the updated information and the balancing device will have to make a decision based on the load, which may differ from the current [8].

The algorithm that will use this information and decide to which server machine to redirect requests, can be implemented in two ways. The first and simplest method is to choose the server that is least busy. It is easy to be implemented, but can lead to imbalances in very high traffic flows and delays in agent-based feedback. The other approach is to use Round Robin balancing with weighting factors, as these factors of the different servers are dynamic and are determined by their load, number of active connections and other static and dynamic values.

III. Load Balancing Implementation

A. Network topology

A minimalistic network topology (Fig. 1) is used in the proposed approach, and it can be further improved based on the business needs. At minimum load balancer itself needs a SDN controller and a SDN compatible switch. The latter will act as a load balancer at the data plane, so all other devices on the network must be connected, either directly or indirectly to the SDN switch. The load balancing SDN switch has many ports, which can be connected to other networks or sub-segments of the same network.

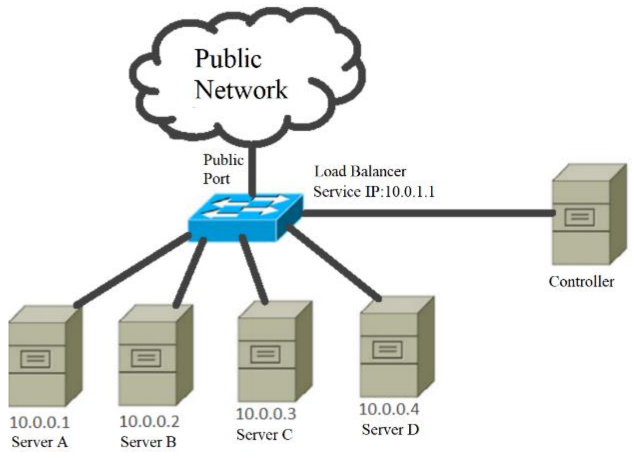

Fig. 1 Simple network topology example

The server machines are connected directly or indirectly to the load balancing switch. These machines all host a service program that is intended to be used by clients through the network. When a request arrives at the load balancing switch it is transmitted to one of these machines. The server machines can be on one or more networks but other machines on these networks cannot access the service through the service IP address. The networks that contain server machines are labeled as internal networks and the load balancer assumes that only server machines that host the service program are on these networks. All other networks are labeled as public networks.

B. Configuration parameters

The implementation of the proposed approach is a program that interacts with the SDN controller. But to function properly it needs to receive several configuration parameters. One of these parameters is the IP address which clients will use to access the service. Another mandatory parameter are the port numbers that connect to the public networks. All other ports are listening on private networks. This is needed because the load balancer can automatically discover all server machines that are on the private networks.

C. Algorithm and implementation

When the program first starts it tries to find and gather information about all hosts on all private networks. This is the so called network topology discovery process which in SDN environment is usually performed based on Link Layer Discovery Protocol (LLDP). In [9], a now method for instant detection of host in SDN networks is proposed which relies on generated ARP-Requests and ARP-Reply packets transmitted by hosts through switch to controller. Thus, in this works this newly suggestion [9] is implemented in the experimental setup. The program first inspects the addresses and network masks on all the switch private ports. In that way, it can compute all the possible IP addresses for the private networks. It then sends an ARP request for each possible IP address and waits for a reply. From the reply to it can then acquire the IP and MAC addresses of all available server machines. The program also keeps track of the servers by sending ARP requests between a specified interval.

The actual load balancing method described below is algorithm agnostic, this means that multiple algorithms such as round robin, or least connecting can be implemented using this method.

When a packet arrives at the load balancing switch that has the destination IP of the service, it is inspected by the controller. Based on the selected load balancing algorithm the controller choses the best possible server machine to handle the requests of this client. An entry with the client and chosen server machine is saved in the controller's memory, and a corresponding rule (flow entry) is added to the switch's forwarding table. To minimize packet loss in case of a server fail, the rule installed on the switch expires much faster than the entry saved on the controller, this allows for the controller to quickly react in case the server machine has failed and at the same time maintaining the speed of the forwarding capabilities of the data plain.

D. Load balancing strategies

In the current implementation only the Round Robin, Least Connection and Dynamic Feedback strategies are used. But any load balancing strategy can be applied using this implementation as a basis.

Unlike the other two strategies, which can function without interacting with the server machines, the Dynamic Feedback strategy relies on constant feedback from each server machine to make its decisions. Thus an agent needs to be installed on each server machine. The agent will send UDP packets with the needed information at a set interval to the load balancing switch. The packet is sent to the service IP and a port preset in the controller configuration. This is achieved by installing a rule on the SDN switch that instructs the switch to send packets with this IP address and port to the controller.

EXPERIMENTS

In Table I the SDN setup description and parameters used for the experiments are given.

TABLE 1
SDN SETUP DESCRIPTION AND PARAMETERS

Description	Tools used
Network Emulation	Mininet
SDN controller type	Centralized
SDN controller	POX
Forwarding element	OpenVSwitch
Flow generator	Curl
Servers load generator and tester	Open load and siege
Number of servers	4
Number of clients	5
HTTP Server	Python Flask

A. Test topology

For the analysis the Mininet emulator is used [10]. The emulated topology is given on Fig. 2.

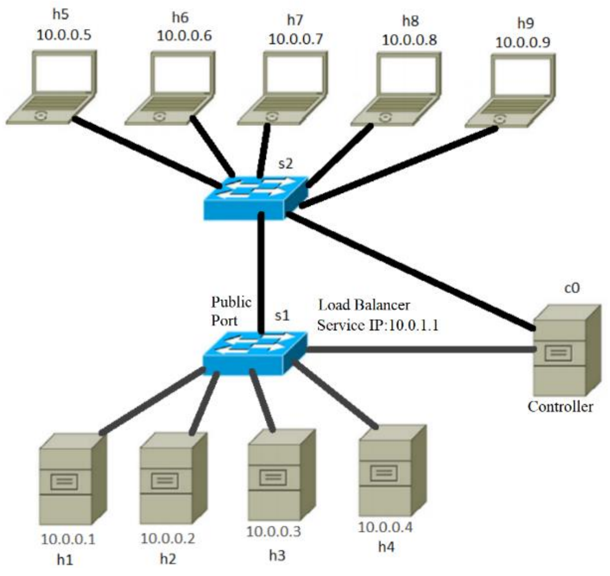

Fig. 2 Test Topology

Nodes h1 through h4 represent the server machines. Nodes h5-h9 represent the clients. The switch s1 is the load balancing switch. The switch s2 acts as a normal layer 2 switch in this scenario.

Each server machine hosts an HTTP server, which will offer a basic web service. Clients will use HTTP client programs such as cURL to send requests to the service.

The HTTP server can work in 3 modes. In the first mode the HTTP server responds to a request without any delay. In the second mode the HTTP server waits a specified number of seconds before responding and in the third mode the HTTP server waits a random number of seconds in a specified interval. The mode is determined by the client's request.

B. Scenario 1

In this experiment, each client will send to the server 200 requests and each request will take a random time (between 5 to 30s) to be processed. Firstly, the Round Robin strategy is used, after that – the Least Connection strategy one. Both approach are compared based on number of connections, average TCP connections, and maximum TCP connection.

TABLE II
ROUND ROBIN STRATEGY – EXPERIMENTAL RESULTS

	h1	**h2**	**h3**	**h4**
Number of connections	250	250	250	250
Avg. TCP connections	46,69	45,8	48,16	46,71
Max. TCP connections	66	66	69	66

TABLE III
LEAST CONNECTION STRATEGY – EXPERIMENTAL RESULTS

	h1	**h2**	**h3**	**h4**
Number of connections	255	242	252	251
Avg. TCP connection	63,34	62,98	62,74	63,15
Max. TCP connections	75	75	74	75

Analyzing the results from Table II and Table III it can be concluded that using the Least Connection strategy will yield a more even spread between the number of TCP connections one server machine can have at a point in time.

C. Scenario 2

The experimental setup is a little bit changed. A UDP agent to monitor server load is added to the HTTP server of the server machines. REST requests are executed, submitting asynchronous tasks for execution, which fictitiously load the server with a given load factor for a certain time. As a result, the HTTP clients can set workload and time for task processing. Each client will send 10 asynchronous request to start a resource consuming operation on the service. As this is an experimental scenario server load is measured by a single coefficient called a weigh coefficient. When an operation starts on the server it increases its weight coefficient by 500 for the next 120 seconds.

TABLE IV
ROUND ROBIN STRATEGY – EXPERIMENTAL RESULTS

	h1	h2	h3	h4
Number of connections	25	25	25	25
Avg. Load on server	113	392	112	110

TABLE V
DYNAMIC FEEDBACK STRATEGY – EXPERIMENTAL RESULTS

	h1	h2	h3	h4
Number of connections	29	13	24	34
Avg. Load on server	127	334	110	150

Analysing the results from Table IV and Table V it can be concluded that dynamic feedback takes the server load into account and can provide better results in such situations. One limitation of this implementation is that the UDP feedback interval is significantly slower than the time the request is coming in. This is why h2 has such a big load.

CONCLUSION

In this paper, an implementation approach for building a load balancer using SDN has been proposed. The implementation has included standard load balancing strategies used in by popular load balancers in production. Experiments have been made comparing the various strategies, their performance on the proposed implementation and in various situation that can occur in practice. To achieve this the Mininet emulator has been used, as well as other technologies like the SDN POX controller for the implementation, the popular Python REST library Flask for the HTTP server and the UDP agent.

ACKNOWLEDGEMENT

The authors would like to thank the Research and Development Sector at the Technical University of Sofia for the financial support.

REFERENCES

[1] P. Göransson, Ch. Black, T. Culver, *Software Defined Networks: A Comprehensive Approach*, Elsevier, 2017.
[2] https://www.opennetworking.org/software-defined-standards/specifications/, last visited: 05.05.2022
[3] K. Nikolova, I. Atanasov, E. Pencheva, “Software defined networks and openflow: A survey”, SGEM’18, Conference Proceedings, Issue 6.3, pp. 697 – 704, Albena, Bulgaria, 2-8 July 2018.
[4] M. Rahman, S. Iqbal, J. Gao, Load Balancer as a Service in Cloud Computing, IEEE 8th International Symposium on Service Oriented System Engineering (SOSE’2014), Conference Proceedings, pp. 204-211, Oxford, UK, 7-11 April 2014.
[5] S. Prabakaran, and R. Ramar, “Software Defined Network: Load Balancing Algorithm Design and Analysis”, International Arab Journal of Information Technology, Vol. 18, No. 3, pp. 312-318, May 2021.
[6] B. Kang, H. Choo, “An SDN-enhanced load-balancing technique in the cloud system”, Journal of Supercomputing, Vol. 74, Issue 11, pp. 5706–5729, Nov. 2018.
[7] S. Kaur, K. Kumar, J. Singh, and N. Ghumman, “Round-Robin Based Load Balancing in Software Defined Networking,” 2nd Intern. Conf. on Computing for Sustainable Global Development, Conference Proceedings, pp. 2136-2139, New Delhi, India, 2015.
[8] X. Yang, H. Shi, Sh. Yang, Zh. Lin, “Load balancing scheduling algorithm for storage system based on state acquisition and dynamic feedback”, IEEE Intern. Conf. on Information and Automation (ICIA’2016), Conference Proceedings, pp. 1737 – 1742, Ningbo, China, 1-3 August 2016.
[9] R. Ch. Meena, M. Nawal, M. Bundele, “Instant Detection of Host in SDN (IDH-SDN)”, International Journal of Recent Technology and Engineering (IJRTE), Vol. 8 Issue 3, pp. 5603-5608, Sept. 2019.
[10] Mininet, http://mininet.org, last visited 05.05.2022